

AD-A081 389

BAERTNER (W W) RESEARCH INC STAMFORD CONN F/S 9/2
PROGRAMMING SUPPORT LIBRARY, VOLUME II. GUIDELINES FOR IMPLEMEN--ETC(U)
NOV 79 C M TURCIO, W M SCHREYER, N A ADAMS F30602-78-C-0103

RADC-TR-79-241-VOL-2 NL

UNCLASSIFIED

1 of 4

AD-A
081389

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

1 of 4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADG-IR-79-241 - Vol II (of two) - 2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROGRAMMING SUPPORT LIBRARY. Volume II. Guidelines for Implementation of Requirements.	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report	
6. AUTHOR(S) Carolyn M. Turcio Wolfgang W. Gaertner / William M. Schreyer Norman A. Adams		7. PERFORMING ORG. REPORT NUMBER N/A
8. PERFORMING ORGANIZATION NAME AND ADDRESS W. W. Gaertner Research, Inc. 205 Saddle Hill Road Stamford CT 06903		9. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0103
10. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441		11. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62728F 25310301
12. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		13. REPORT DATE November 1979
14. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		15. NUMBER OF PAGES 322
16. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		17. SECURITY CLASS. (of this report) UNCLASSIFIED
18. SUPPLEMENTARY NOTES RADG Project Engineer: Michael Landes (ISIE)		19. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
20. KEY WORDS (Continue on reverse side if necessary and identify by block number) Programming Support Library Software Engineering Structured Programming Software Development Management		
21. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report, Volume II, contains implementation guidelines for each paragraph in Volume I with examples drawn from several different types of computer systems ranging from minicomputers to large in-house mainframes and time-sharing systems. The examples serve to further illustrate the intent of each requirement and will aid in building whatever new software is required to perform the required function.		

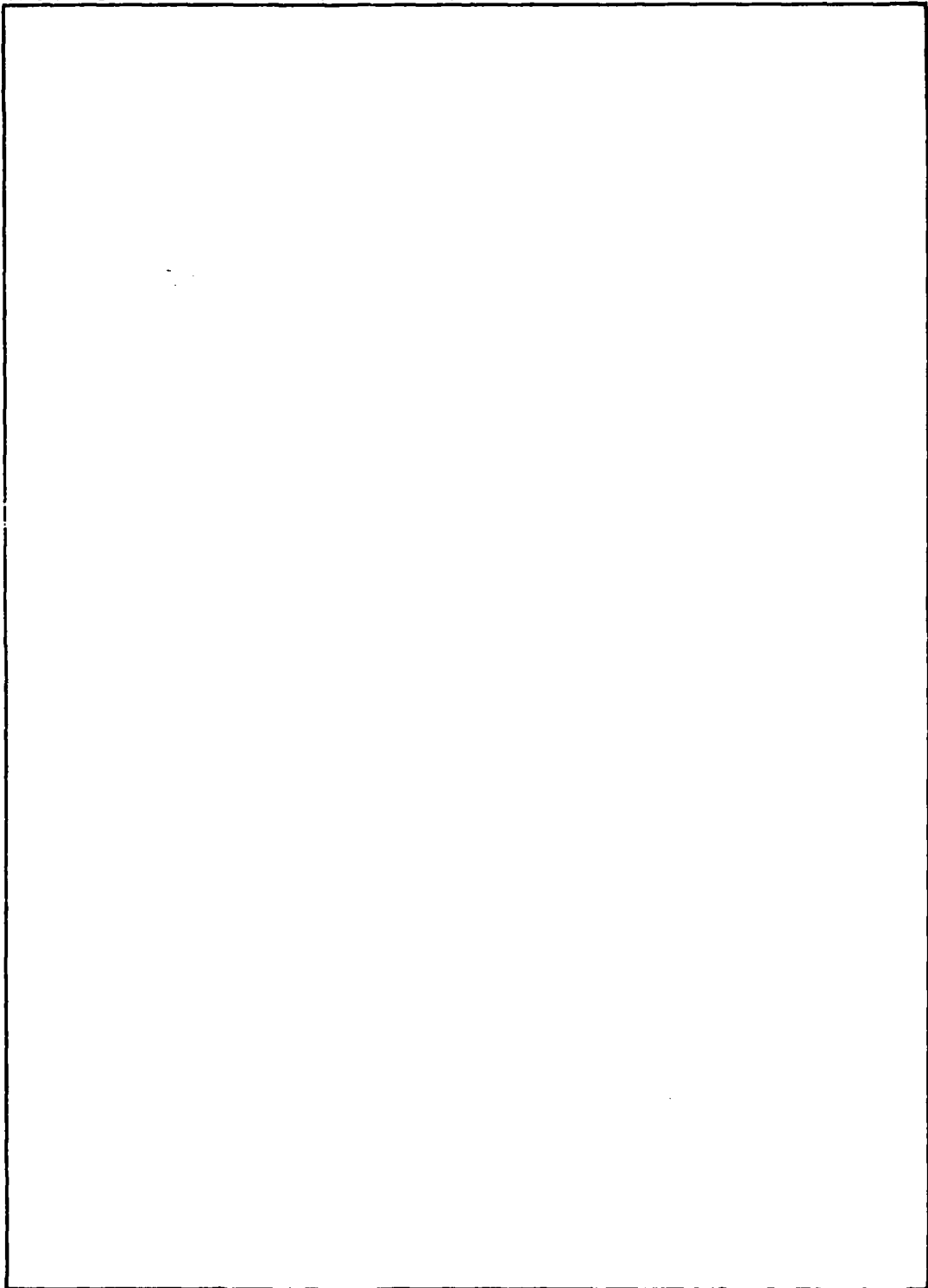
DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

Chapter No.	Page No.
1. INTRODUCTION	1
2. FORMAT OF GUIDELINES FOR IMPLEMENTATION	2
3. GUIDELINES FOR IMPLEMENTATION OF REQUIREMENTS	6
3.1 Source Data Maintenance	6
3.1.1 Basic Source Data Maintenance	7
3.1.1.a Data File Storage	7
3.1.1.b Data Access	16
3.1.1.c Library Backup Capability	19
3.1.1.d Data Maintenance	27
3.1.2 Full Source Data Maintenance Requirements	74
3.1.2.a Data File Storage	74
3.1.2.b Full Data Maintenance	79
3.1.3 Summary of Facilities for Source Data Maintenance	95
3.2 Output Processing	96
3.2.1 Basic Output Processing Requirements	96
3.2.1.a Library Control Listings	96
3.2.1.b Source Data Listings	103
3.2.1.c Control Data	107
3.2.2 Full Output Processing Requirements	112
3.2.2.a Full Magnetic Tape Output	112
3.2.2.b Full Punched Card Output	117
3.2.2.c Full Management Control Listings	119
3.2.2.d Full File Directory and Listings by Programmer	119
3.2.2.e Full Automatic Indentation of Source Code Listings	125
3.2.2.f Full Data Scanning Capability	127
3.2.2.g Full Directory Availability Printout	130
3.2.3 Summary of Facilities for Output Processing	132
3.3 Programming Language Support	133
3.3.1 Basic Programming Language Support Requirements	133
3.3.1.a Compiler Interface	133
3.3.1.b Load Module Generation	138
3.3.2 Full Programming Language Support Requirements	143
3.3.2.a Full Top Down Structured Programming Support	143
3.3.3 Summary of Facilities for Programming Language Support	158

Accession For	1	Distribution/	Availability Codes	Available for Special
	A			

3.4	Library System Maintenance	159
3.4.1	Basic Library System Maintenance	159
3.4.1.a	Library Installation Support	159
3.4.1.b	Library Maintenance Support	163
3.4.1.c	Library Termination Support	168
3.4.2	Full Library System Maintenance Requirements	172
3.4.2.a	Full Library Installation Support	172
3.4.3	Summary of Facilities for Library System Maintenance	174
3.5	Data Security	177
3.5.1	Basic Data Security Requirements	177
3.5.2	Full Data Security Requirements	185
3.5.2.a	Full Data Integrity	185
3.5.2.b	Full Data Protection	190
3.5.3	Summary of Facilities for Data Security and Classified Work	197
3.6	Management Data Collection and Reporting	198
3.6.1	Basic Management Data Collection and Reporting Requirements	198
3.6.2	Full Management Data Collection and Reporting Requirements	198
3.6.2.a	Full Collecting	199
3.6.2.b	Full Updating	203
3.6.2.c	Full Accumulating	204
3.6.2.d	Full Archiving	205
3.6.2.e	Full Reporting	206
3.6.3	Additional Suggested Specifications for Management Data Collection and Reporting Programs and Data Structures	209
3.6.3.1	Data Structures	209
3.6.3.2	Software Specifications	215
3.6.3.2.1	Update	215
3.6.3.2.2	MDFILE	227
3.6.3.2.3	MDUPDATE	236
3.6.3.2.4	MDREPORT	239
3.6.3.2.5	MDAUTORPT	242
3.6.3.2.6	MDPRINT	245
3.6.3.2.7	MDHIST	249
3.6.3.2.8	MDXCHECK	252
3.7	Documentation Support	253
3.7.1	Basic Documentation Support Requirements	253
3.7.2	Full Documentation Support Requirements	254
3.7.3	Comparative Analysis of Documentation Support Facilities	266
3.8	On-Line Terminal Implementation Requirements	277
3.8.1	Basic On-Line Terminal Implementation Requirements	278
3.8.2	Full On-Line Terminal Implementation Requirements	295

3.8.3	Summary of On-Line Terminal Capabilities	304
3.9	General Requirements	305
3.9.1	Basic General Requirements	305
3.9.2	Full General Requirements	305
3.9.2.a	Concatenation of Data Files	306
3.9.2.b	Subroutine Support for User Programs	307
3.9.3	Summary of General Requirements	308
4.	SUMMARY AND CONCLUSIONS	308
4.1	Basic PSL Implementation	308
4.1.1	Type I System Summary	308
4.1.2	Type II System Summary	311
4.1.3	Type III System Summary	311
4.2	Full PSL Implementation	

1. INTRODUCTION

This Report on the requirements for a Programming Support Library is organized into two volumes.

Volume I contains the Functional Requirements and Volume II contains the Guidelines for their Implementation including many examples generated on a variety of computer systems.

In Volume I the Functional Requirements are presented at two different levels of detail: Chapter 2 contains a brief summary of the requirements, separated into the requirements for a Basic PSL (Chapter 2.1) and a Full PSL (Chapter 2.2).

Chapter 3 contains the full wording of the functional requirements organized into 9 different categories. The specification for each individual requirement is followed by a brief description of the purpose of the particular paragraph and its classification (Basic or Full).

Volume I of this Final Report may therefore be used as the specification for the implementation of a Basic or Full Programming Support Library (PSL) under a Government software development contract.

This Volume II of the Final Technical Report follows the organizational structure of Volume I and provides implementation guidelines for each paragraph, with examples drawn from several different types of computer systems ranging from minicomputers to large in-house mainframes and time-share systems. The examples serve as further illustrations of the intent of each PSL requirement and can therefore be used by a DP manager to rapidly assess whether his present installation already complies with a specific PSL requirement. If not, the guidelines and examples will allow him to implement a requirement in a cost effective manner.

All references to the "Structured Programmed Series" pertain to RADC-TR-74-300 which consists of 15 volumes. It is available from the NTIS.

2. FORMAT OF GUIDELINES FOR IMPLEMENTATION

PSL requirements have been divided into nine major functional areas. Many of these areas in turn are subdivided into two types of requirements: Basic and Full. Basic requirements are those which either form the minimum needed to provide a PSL or which are fulfilled by facilities so common to all computer installations as to be considered a basic part of standard systems. Full requirements generally necessitate additional time and space, but when met provide a PSL of greatest effectiveness for all personnel involved in the software development effort. These full requirements are most appropriate for installations involved in system development and/or large scale applications programming.

It is the purpose of this report to show that a Basic PSL can be implemented with currently existing facilities, and usually does not require additional cost on the part of the contractor. On the other hand, the implementation of some of the Full PSL requirements will require considerable effort, as described throughout this report and summarized in Chapter 4. The types of computer installations most commonly used today may be broken down into three main categories:

Type I - Minicomputer system existing completely in-house. This type of installation is primarily interactive in nature, and often is a single-user facility. Typical examples of this are systems built around a PDP-11, NOVA, or Hewlett-Packard 21MXE.

Type II - Commercial time sharing facilities. This type of installation is also primarily interactive in nature, but has the advantage of permitting multiple concurrent users. Typical examples are IBM's Conversational Monitoring System, National CSS, and those systems supported by Data General's Eclipse under AOS.

Type III - Large computer systems operating primarily in batch mode but having some limited interactive capabilities. A typical example of this type of system is the CDC 6600.

The examples in this report will be presented in terms of these three main categories of installations. Discussions will also include evaluations of comparative merit of each type as it meets the requirement of a specific PSL function.

In discussing the organization of PSLs, the following terms have been used:

Character: The most basic unit of data. It was decided to present PSL contents in terms of characters since most PSL data (e.g., program source code and object code, job control data, textual data) are viewed as character strings. Even raw data when analyzed for debugging is seen in that light. However, there is always a byte-to-character ratio for any given system, allowing ready transition from the character-oriented to the byte-oriented viewpoint.

Record: A collection of data, i.e., a sequence of characters. Record length varies with context, but basically a record is that unit of data which may be manipulated as a coherent entity at any one time. In a batch-oriented system, a record would be 80 characters long, i.e., the contents of one punched card. In an interactive environment, a record would be equivalent to a line of data as it is accessed and manipulated by standard software tools such as an editor. While not all data sets may be viewed in terms of records (e.g., relational data bases form an exception), the concept of a record is clearly enough understood throughout the data processing community so as to make it a convenient measure for data.

File: A collection of one or more related records which form an organized body of information which may be classified as a specific Data Type. Files are generally stored as coherent units that are accessible by a name specified either by the user or the system.

Library: A collection of files which are related in terms of nature or usage. For example, all source modules that are currently active in a given system may be stored in the source module library, while the object code resultant from translations would reside in the object module library. Alternatively, libraries may be organized around individual projects, such that all source, object and load modules for one project, as well as the documentation and accounting, may form a single library.

Data Types: A classification by function of the types of data which may be stored and/or used. These are:

1. Program source code.
2. Program object modules.
3. Program load modules.
4. Job control data.
5. Test data.
6. Program Design Language statements.
7. Textual data.
8. Other data.

Chapter 3 which contains the guidelines for the implementation of the Functional Requirements follows a specific systematic format as follows:

Title of requirement;

Classification (Basic or Full);

Full text of requirement;

Purpose of requirement;

Description of existing tools (programs) which satisfy this requirement;

Examples from various types of computer systems;

Program requirements and definitions of new support programs needed to satisfy the PSL requirement;

Suggestions for efficient implementation of requirement.

Wherever applicable, the above format is followed throughout Chapter 3 of this report.

The report closes with a general summary and a statement of conclusions.

References to a specific volume of the Structured Programming Series (RADC TR-74-300) are abbreviated as SPS V (Structured Programming Series, Volume V).

3. GUIDELINES FOR IMPLEMENTATION OF REQUIREMENTS

3.1 Source Data Maintenance

Requirement:

This functional area addresses the physical storage and maintenance of programming related data. The basic requirements define the minimum capabilities needed to store, access, maintain and provide backup for the data. The full requirements define additional capabilities which increase storage efficiency and improve maintenance support.

Purpose of Paragraph:

To define the nature and scope of source data maintenance.

Existing Tools Satisfying this Requirement:

Storage and backup of data is normally available by accessing (a) specific programs (e.g., IBM's UTILITIES package), (b) specific system components (e.g., DEC's Peripheral Interchange Program), or (c) commands which are an integral part of an interactive system (e.g., IBM's CMS commands of COPY and TAPE).

Access to and maintenance of source data files is done via an interactive editor or a set of editing programs. Some such form of editing facility is standard to even the most basic installation.

Further examples may be found under specific headings which follow.

3.1.1 Basic Source Data Maintenance

3.1.1.a Data File Storage

Classification: Basic

Requirement:

Physical storage must be provided for the following types of data.

1. Program source code - Programmer coded input to a program language compiler or assembler (e.g., COBOL source statements).
2. Program object modules - Program code that results from the execution of a compiler or assembler.
3. Program load modules - Program code in a form ready to be loaded into a computer for execution.
4. Job control data - Control statements used to control the execution of computer jobs.
5. Test data - Data used for testing programs and/or program systems during the development and maintenance of the program/system and for performing system and acceptance testing.
6. Program Design Language (PDL) statements - English-like statements that follow the basic rules of structured programming and are used to define the program structure and logic.
7. Textual data - Standard text which is written primarily by programmers for use as program documentation. (Although the intent is to support program documentation, any textual data supporting the programming process could be stored and maintained in the PSL.)
8. Other data - Any form of data that can be entered via punched cards, magnetic tape, or on-line terminals and stored in a computer. What data is to be stored is determined by the user of the system.

Purpose of Paragraph:

To define types of data files requiring physical storage.

Examples:

Example 3.1.1.a.1: Storage of data in a Type I system.

In this single-user system, storage of the various file types is accomplished as follows:

- (1) Program source code is created using the text editor, and stored under the user-specified name upon exit from the editor.
- (2) Program object modules are generated upon successful compilation/assembly, and are assigned an extension name (3 letter name following the ".") of OBJ.
- (3) Program load modules are created when the linker has successfully executed. These load modules are given an extension type of .SAV, .REL, or .LDA, depending on the kind of memory image the user requests.
- (4) Job control sequences may be stored in BATCH files (extension name .BAT) which can process one or more programs.
- (5) Test data, Program Design Language statements, and textual data may be created by the editor. Test data is usually assigned the extension name .DAT.
- (6) Card files may be entered via the reader and stored on the user's disk as shown below. Storage on tape is also possible.

Example 3.1.1.a.2: Storage of data in a Type II system.

In a Type II system, data is stored on disk(s) allocated to individual virtual machines. The storage process may occur in the following ways:

1. Program source code is created by entering the editor and entering a file consisting of source code statements in an installation-supported language. Actual storage occurs when the editor command FILE is entered. Filing occurs under the name designated by the user.

```

.R EDIT
*EWPROG1.FOISS
*IC      SAMPLE PROGRAM
C
      READ(5,1000) V,M
1000     FORMAT(2I4)
      L = V**M
C
      WRITE(6,1001) V,M,L
1001     FORMAT(1X,I4,4I **,I4,3I = ,I10)
      STOP
      END
SE$$$

```

```

.R FORTRAN
*PROG1.OBJ, PROG1.LST=PROG1.FOR
*IC

```

```

.R PIP
*PROG1.* /L
21-MAR-73
PROG1 .FOR      1 21-MAR-73
PROG1 .OBJ      6 21-MAR-73
PROG1 .LST      2 21-MAR-73
3 FILES, 9 BLOCKS
3027 FREE BLOCKS
*IC

```

```

.R LINK
*PROG1.SAV=PROG1.OBJ/
*IC

```

```

.R PIP
*PROG1.* /L
21-MAR-73
PROG1 .SAV     13 21-MAR-73
PROG1 .FOR      1 21-MAR-73
PROG1 .OBJ      6 21-MAR-73
PROG1 .LST      2 21-MAR-73
4 FILES, 22 BLOCKS
3014 FREE BLOCKS
*FTV5.DAT=CR:
*TT:=FTV5.DAT
  2    5
*IC

```

```

.R EDIT
*EWPROG1.BAT$$
*ISJOB/RT11
.ASSIGN RK 5
.ASSIGN TT 6
.R PROG1
SE$$$

```

Figure 3.1.1.a-1. Storage of Data in a Type I System.

.ASSIGN RK LOG

.ASSIGN RK LST

.LOAD BA

.R BATCH

*PROG1

END BATCH

.R PIP

*TT:=PROG1.LOG

\$JOB/RT11

2 ** 5 = 32

STOP --

\$EOJ

*! C

Figure 3.1.1.a-1: (Continued) - Storage of Data in a Type I System.

2. Program object modules are created and automatically stored onto disk with a file type of TEXT following successful compilation/assembly of the source code. File name remains the same as the source file.
3. Program load modules are created with the LOAD command (TEXT file arguments). The modules may then be stored by issuing the GENMOD command, thus creating a file of type MODULE.
4. Job control statements may be stored in files with a file type of EXEC, which are created via the editor. These control statements may then be executed by issuance (as a command) of the file name associated with the EXEC file in which they reside.
5. Test data, Program Design Language statements and textual data may all be created as files via the editor.
6. CMS provides each virtual machine with a read buffer which allows punch card images or files to be sent to the machine from a card reader or other machine, then stored from buffer to disk with the READCARD command. In CSS, the OFFLINE READ command accomplishes a similar purpose.
7. Interactive system formatted files may be stored onto tape or restored from tape with the TAPE command.
8. Any file may be transferred from one virtual machine to another by the use of the SPOOL PUNCH and PUNCH commands in CMS, or by the XFER command in CSS.

The following example shows the CSS storage methods for all but items 7 and 8 above, which are illustrated elsewhere. Note that the stack created by the &STACK command in the EXEC file functions to stack terminal data input to the programs which the preceding EXEC file commands execute.

```

21.02.12 >edit prog1 fortran p
NEW FILE.
INPUT:
>c sample program
>c
>      read(5,1000) n,m
>1000  format(2i4)
>      l = n ** m
>c
>      write(6,1001) n,m,l
> 1001 format(' ',i4,' ** ',i4,' = ',i10)
>      stop
>      end
>
EDIT:
>file

```

```

21.03.58 >fortran prog1
FORTRAN G1 2.0

```

```

21.04.32 >listf prog1
FILENAME FILETYPE MODE      ITEMS
PROG1    FORTRAN    P        10
PROG1    TEXT       P        17

```

```

21.04.46 >load prog1

```

```

21.04.55 >genmod prog1

```

```

21.05.00 >listf prog1
FILENAME FILETYPE MODE      ITEMS
PROG1    FORTRAN    P        10
PROG1    TEXT       P        17
PROG1    MODULE     P         3

```

```

21.05.55 >edit sample exec p
NEW FILE.
INPUT:
>&stack 2,7,
>load prog1 (xeq)
>
EDIT:
>file

```

```

21.06.32 >sample
21.06.35 LOAD PROG1 (XEQ)
EXECUTION:

```

```

      2 **      7 =      128

```

```

21.06.38 >xfer punch to adams

```

```

21.06.41 >offline punch newfile data p

```

```

21.06.42 >xfer punch off

```

Figure 3.1.1.a-2. Storage of data in a Type II system.

Example 3.1.1.a.3: Storage of data in a Type III System.

Since Type III systems are batch oriented with only small interactive capability, the main mode of storage is on punch cards. Unless specifically requested, object and load modules are created only for the duration of a job, during which time they reside on a system disk. Job control statements and data as well as source code all are on cards.

The interactive facility allows storage of the File Types as follows:

1. Program source code is created with the editor in the CREATE mode, then saved as a local file with the SAVE command. Local files are temporary, remaining only as long as the user is logged on. Local file may be made permanent via the CATALOG or STORE commands.
2. Object modules are created upon compilation of source programs. The module is called LGO and will automatically be overwritten by the next RUN command unless it is permanently saved under another name. Re-execution would then require the saved file to be copied under the name LGO by entering the editor, loading it, and running the SAVE, LGO command.
3. This system does not provide a means of permanently saving the load modules.
4. This system does not provide any special means for storing job control statements other than as a standard data file which may be referenced for documentation (i.e., the user cannot use that file to execute the program).
5. Test data (as shown in the example below); Program Design Language statements, and textual data may all be created with the editor and stored as discussed in 1) above.
6. Card images may be sent to a terminal from its associated reader and stored under the name designated by the user with the READ command. Magnetic tape and paper tape input are also possible.

7. This system does not allow storage of interactively created or stored files on any media other than on the disk associated with the interactive facility on paper tape, and on magnetic tape.
8. This system does not provide a means of transferring data files directly from one user's terminal to another.

The following example shows how a user creates a source file, permanently stores it, creates the data file and temporarily stores it. Also shown are the creation (via compilation execution) of the object module and its storage, and examination of the output file as well as permanent storage of files sent from a card reader.

Suggestions for Efficient Implementation:

On-line storage (core and/or disk) is highly advantageous for frequently accessed data, whereas data used less often may be stored on mountable volumes, such as tape or floppy disk without significant loss. This division of storage media according to access frequency reduces the physical storage costs, while the use of interactive mode to manipulate key data increases effective utilization of human resources.

Ease of use is greatly facilitated by a system which is designed as primarily interactive, whether it is single- or multi-user. Of the three system types examined, only the Type III (primarily batch) did not already have facilities to support storage of all PSL data types. The revisions needed to bring this type of system up to the standard would be extensive.


```

COMMAND-  EDITOR
..C
100=      PROGRAM SAMPLE (TAPE5, TAPE6)
110= C    SAMPLE PROGRAM CALCULATES N**M
120= C
130=      REWIND 5
140=      READ (5, 700) N, M
150= 700  FORMAT (2F4)
160=      L=N**M
170=      WRITE (6, 800)
180= 800  FORMAT (F4,'**',F4,'=',F10)
190=      STOP
200=      END
210= =
..SAVE, S1, N
..BYE
COMMAND-  CATALOG, S1, PROG1, ID=GAERTNER
NEW CLCYE CATALOG
RP= 090 DAYS
CT ID=GAERTNER PFN=PROG1
CY ID=002
COMMAND-  EDITOR
COMMAND-  FETCH, PROG1, GAERTNER
..CREATE, SUP
ENTER LINES
    2; 7
=
..SAVE, TAPE5, N
..FILES
--LOCAL FILES --
TAPE5 *PROG1
..RUN, FTN
    .172 CP SECONDS COMPILATION TIME
    STOP
    .036 CP SECONDS EXECUTION TIME
..FILES
TAPE5 *PROG1 TAPE6 &OUTPUT &INPUT
LGO
..EDIT, LGO, NOSEQ
..SAVE, PROG1LD
..EDIT, TAPE6, SEQ
..LIST, ALL, SUP
    2**7=128
..BYE
COMMAND-  CATALOG, PROG1LD, PROG1LD, ID=GAERTNER
NEW CYCLE CATALOG
RP= 090 DAYS

```

Figure 3.1.1.a-3. Storage of data in a Type III system.

CT ID=GAERTNER PFN=PROG1LD
CT CY=003 00000128 WORDS.:
COMMAND- READ,NEWFILE
COMMAND- CATALOG, NEWFILE,NEWFILE, ID=GAERTNER

Figure 3.1.1.a-3 (continued). Storage of data in a Type III system.

3.1.1.b Data Access

Classification: Basic

Requirement:

Direct access of a single record of stored data must be provided via an on-line storage device when access to that data is deemed necessary to the user. Such access is necessary during execution of standard PSL jobs (e.g., data file updating or other maintenance operations).

Purpose of Paragraph:

To emphasize the need for immediate availability of important files and portions thereof so as to facilitate job analysis and error correction procedures, if any.

Existing Tools Which Satisfy This Requirement:

An editor which allows access to and interaction with file contents is the prime tool which satisfies this requirement.

Examples:

Example 3.1.1.b.1: Accessing a record of data in a Type I system.

The typical in-house minicomputer system includes an editor which can access any type of data file - whether source code, text, or data - and can manipulate it per user commands. In the example below, the source data file created in the preceding example is made available to the editor, and its third record is accessed by moving the pointer past two end-of-line (carriage return, Line Feed) markers, and is displayed by the Verify command.

```
.R EDIT
*EBPROG1.FOR$1$2A$V$3
      READ(5,1000) N,M
*
```

Figure 3.1.1.b-1. Accessing a record of data in a Type I system.

Example 3.1.1.b.2: Accessing a record of data in a Type II System.

Commercial time-sharing systems offer the easiest access of the three system types considered. In the sample dialogue below, the third record of the previously created source program is accessed by advancing the pointer down three lines (records). Notice that the initial pointer position is at record 0., whereas the pointer in the Type I system was initially positioned immediately in front of the first character of the first record. In this Type II system verification is automatic.

```
21.57.44 >edit prog1 fortran
EDIT:
>down 3
      READ(5,1000) N,M
>
```

Figure 3.1.1.b-2. Accessing a record of data using a Type II system.

Example 3.1.1.b.3: Accessing a record of data in a Type III system.

As in the other two systems, the typical Type III editor can handle files of all data types. The example following shows access of the second record of the previously created source file, but a data or text file could be as easily accessed. Already existing files must be accessed via the ATTACH or FETCH commands. Individual records therein may be accessed with the editor, by specifying the line number of the desired record.

```
COMMAND - ATTACH, S1 PROG1, ID=GAERTNER
          PF CYCLE NO. = 002
COMMAND- EDITOR
..E, S1, S
..L, 110
120=C    SAMPLE PROGRAM CALCULATES N**M
```

Figure 3.1.1.b-3. Accessing a record of data in a Type III system.

3.1.1.c Library Backup Capability

Classification: Basic

Requirement:

A capability must be provided to recover from inadvertent loss or destruction of data. This involves the following two functions.

1. Backup storage on a storage unit independent of the master file (e.g., magnetic tape, disk pack, punched cards).
2. Regeneration of the library data files from backup storage.

The backup facility must provide the capability to selectively generate and restore backup data so that it is possible to recover portions of the total PSL data base without the need to perform full storage dump and restore operations.

Purpose of Paragraph:

To define the functional requirements which allow recovery of lost or destroyed data.

Existing Tools Which Satisfy This Requirement:

- a) Hardware - Facilities which allow writing to and reading from at least one of:

Paper tape, cassette tape, floppy disk, magnetic tape, disk pack, or punched cards.
- b) Software - Any program or set of programs which allows the user to cause a storage medium to be physically attached to the main system and to direct selective information transfer between the system and that medium. The software should also make provisions for the internal labelling of the storage medium. IBM's UTILITIES package fulfills this requirement as does DEC's Peripheral Interchange Program and certain utility command within National CSS time-share system, among others.

Examples:

a) Backup of data

Backup of PSL files onto one of the media previously listed is most easily achieved using an interactive system implemented either on a small in-house computer (Figure 3.1.1.c-1) or large-scale time-share computer (Figure 3.1.1.c-2). While backup processing is more cumbersome using a Type III, system, it is still feasible.

Example 3.1.1.c.1: Backup of disk files onto magnetic tape using a Type I system.

Like most minicomputer systems, RT-11 (the Type I representative) has a file handling program which is a standard utility. It is called Peripheral Interchange Program (PIP), and enables full file manipulation among the various storage media.

The example below first activates PIP, then specifies that all files with an extension designation of .FIG (*.FIG) should be individually copied (/X) from the default device (user disk) to magnetic tape device MT0. The files are to retain the same names. The listing of the contents of the tape verifies correct transfer. Several other files are likewise transferred, the /M command followed by a positive integer specifying that the tape should not be rewound before writing the files. Finally, the contents of the entire tape are listed.

Example 3.1.1.c.2: Backup of disk files onto magnetic tape using a Type II system.

In the following example, the time of day followed by a "greater than" sign provides the CSS system prompt that indicates that the user is in command mode. In the top line the user requests the operator to mount the tape with the label X2234 onto the device (a tape drive) with address 282, leaving the ring in. The operator then replies that the device is attached, whereupon the user issues an abbreviated form of the TAPE REWIND command. While this step is not actually necessary, it serves as a good means to ensure physical mounting of the tape by the operator, and is an advisable precaution in all time-sharing systems.

```

.R PIP
*MTB:*.***.FIG/X
*MTB:/L
18-NOV-76
B55T01.FIG      1 18-NOV-76
B55T02.FIG      1 18-NOV-76
B55T03.FIG      1 18-NOV-76
3 FILES, 3 BLOCKS
*MTB:B55T4T.FOR=B55T4T.FOR
*MTB:B55CS2.***B55CS2.*/X/M:1
*MTB:B55GEN.***B55GEN.*/X/M:1
*MTB:*.***.SAM/X/M:2000
*MTB:/L
18-NOV-76
B55T01.FIG      1 18-NOV-76
B55T02.FIG      1 18-NOV-76
B55T03.FIG      1 18-NOV-76
B55T4T.FOR      1 18-NOV-76
B55CS2.SAV      12 18-NOV-76
B55GEN.SAV      15 18-NOV-76
B55GEN.FOR      1 18-NOV-76
B55EBS.SAM      3 18-NOV-76
B55EBB.SAM      3 18-NOV-76
B55CBB.SAM      3 18-NOV-76
BXTMP.SAM       1 18-NOV-76
B55SS7.SAM      3 18-NOV-76
B55SS8.SAM      2 18-NOV-76
B55SS9.SAM      1 18-NOV-76
B55CD1.SAM      2 18-NOV-76
B55CD2.SAM      3 18-NOV-76
B55CD3.SAM      3 18-NOV-76
B55CD5.SAM      1 18-NOV-76
B55CD6.SAM      2 18-NOV-76
B55CD7.SAM      3 18-NOV-76
B55CD8.SAM      2 18-NOV-76
B55CD9.SAM      1 18-NOV-76
B55CDA.SAM      2 18-NOV-76
23 FILES, 67 BLOCKS

```

Figure 3.1.1.c-1. Backing up disk files on magnetic tape using a Type I system.

The user then requests that the first 16 files be skipped and the remainder of the tape be scanned. The system responds with the file name and file type of all files on the remainder of the tape. The user then requests transfer of all files with a file type of DATA from his disk to the mounted tape. The computer responds with a confirmation of the current process and the label of each file as it is transferred.

Finally, a tape mark denoting end-of-file is written, another single file is transferred and followed by a tape mark, and the user requests the tape device to be detached.

Note that in this typical interactive time share system, the user needed to know only three commands, their abbreviations, and parameters in order to perform backup. The operation itself took less than nine minutes, most of which was spent waiting for the physical mounting of the tape.

Example 3.1.1.c.3: Backup of disk files onto magnetic tape using a Type III system.

Backup can only be accomplished by means of the COPY utility, submitted either in the form of a card deck for standard batch, or as an interactive file sent to the central site via the BATCH command. In either case, the COPY job file would appear as shown in Figure 3.1.1.c-3. The job card indicates job name, estimated CPU time, estimated input/output time, and other pertinent job parameters. The 0 following input/output time is used to request an infinite time limit. Physical devices and files to be used in the job are then denoted via the REQUEST cards. Finally the COPY utility is requested to copy the named permanent files onto tape. Note that the user must individually designate each file to be transferred to tape.

14-01-32 >MOUNT TAPE X2234 AS 282 RINGIN

14-02-01 >DEV 282 ATTACHED
>T REW

14-07-22 >TAPE SKIP 16

14-08-39 >TAPE SCAN

SWITCHES FORTRAN

SW37A DATA

SW38 DATA

SW3720 DATA

SW3738 DATA

SW372 DATA

SW3713 DATA

* EOF

14-08-44 >TAPE DUMP * DATA
DUMPING...

SW37B DATA

SW38B DATA

SW38C DATA

SW3738B DATA

SW374B DATA

SW374C DATA

14-09-05 >TAPE WTM

14-09-16 >TAPE DUMP SWNVERB FORTRAN

14-10-02 >TAPE WTM

14-10-13 >DET 282

Figure 3.1.1.c-2. Backing up disk files on magnetic tape using a Type II system.

```
BKUP, T30, 100, MT1.  
REQUEST, TAPE, MT.  
VSN, TAPE, 1492.  
REQUEST, B55DATA, A*.  
REQUEST, B55PGM, A*.  
REQUEST, S41RANDOM, A*.  
REQUEST, S41GEN, A*.  
COPY, B55DATA, TAPE.  
COPY, B55PGM, TAPE.  
COPY, S41RANDOM, TAPE.  
COPY, S41GEN, TAPE.
```

Figure 3.1.1.c-3. Backup of disk files on magnetic tape using a Type III system.

b) Recovery of Data

Recovery of data which has been obtained from the same system is usually an inverse of the backup procedure, and often uses the same commands. Following are two examples of recovery procedures using an in-house computer (Figure 3.1.1.c-4) and a time sharing system (Figure 3.1.1.c-5).

Example 3.1.1.c.4: Recovery of files from magnetic tape using a minicomputer.

This illustrates the recovery of data backed up in Figure 3.1.1.c-1. Upon entering PIP, the user requests a listing of the tape contents.

These contents are then restored as desired to the user disk (default option, therefore not specified) from MT0. Note that not all tape contents need to be retrieved at once; they can be selectively restored.

```

.R PIP
*MT0:/L
18-NOV-76
B55T01.FIG      1 18-NOV-76
B55T02.FIG      1 18-NOV-76
B55T03.FIG      1 18-NOV-76
B55T4T.FOR      1 18-NOV-76
B55CS2.SAV     12 18-NOV-76
B55GEN.SAV     15 18-NOV-76
B55GEN.FOR      1 18-NOV-76
B55ED5.SAM      3 18-NOV-76
B55ED6.SAM      3 18-NOV-76
B55CDE.SAM      3 18-NOV-76
BX'MP'.SAM      1 18-NOV-76
B55SS7.SAM      3 18-NOV-76
B55SS8.SAM      2 18-NOV-76
B55SS9.SAM      1 18-NOV-76
B55CD1.SAM      2 18-NOV-76
B55CD2.SAM      3 18-NOV-76
B55CD3.SAM      3 18-NOV-76
B55CD5.SAM      1 18-NOV-76
B55CD6.SAM      2 18-NOV-76
B55CD7.SAM      3 18-NOV-76
B55CD8.SAM      2 18-NOV-76
B55CD9.SAM      1 18-NOV-76
B55CDA.SAM      2 18-NOV-76
23 FILES, 67 BLOCKS
***=MT0:*.FIG/X
*B55GNB.FOR=MT0:B55GEN.FOR
*B55GNB.* /L
18-NOV-76
B55GNB.FOR      1 18-NOV-76
582 FREE BLOCKS
*
```

Figure 3.1.1.c-4. Recovery of files from magnetic tape on a Type I system.

Example 3.1.1.c.5: Recovery of files from a magnetic tape using a Type II System.

This illustrates the recovery of data backed up in Example 3.1.1.c.2. Once again the user requests the mounting of the desired tape, and upon receiving confirmation that the tape drive has been given to him, he then proceeds to position the tape at the beginning. He then skips forward an approximate number of files, and requests that a specific file be transferred from tape to disk. Files on tape are scanned until the correct file is found, and it is then copied. The user confirms correct transfer by requesting that the entry in the disk directory for that file be displayed. The tape mark following this file is skipped via the SKIP command, then the user requests the remainder of the tape to be loaded. The system lists file name and file type of the file transferred. The user then confirms that the file has been properly entered in the disk directory and finally requests the tape drive be logically detached.

14.14.23 >MOUNT TAPE X2234 AS 282

14.15.03 >DEV 282 ATTACHED
>TAPE REWIND

14.20.14 >TAPE SKIP 17

14.21.39 >TAPE LOAD SWN374C DATA

14.22.39 >LISTF SWN374C DATA
SWN374C DATA P 53

14.22.52 >TAPE SKIP

14.23.07 >TAPE LOAD * *
LOADING...
SWNVERB FORTRAN
* EOF

14.23.49 >LISTF SWNVERB FORTRAN
SWNVERB FORTRAN P 235

14.24.20 >DET 282

Figure 3.1.1.c-5. Recovery of files from magnetic tape on a Type II system.

3.1.1.d Data Maintenance

Classification: Basic

Requirement:

The PSL must provide facilities for generation and updating of library data files. Updating shall be defined to include addition to, deletion from, replacement within and copying of a file or parts thereof, as well as organization of file references to reflect such changes. Specifically, the minimum capability required includes the following functions:

Purpose of Paragraph:

To state the minimum requirements for data maintenance, and to fully define those requirements.

Existing Tools Which Satisfy this Requirement:

Batch systems include utility routines which allow users to generate and/or alter data. Interactive systems can have editors which facilitate such data manipulation. However, editors vary widely as to both approach taken and scope and flexibility of commands. The following is a functional summary of three types of editors.

I. Functional capabilities of Type I editor

1. Modification of pointer position.
 - a) To top line of file (BEGINNING)
 - b) Forward or backward a specified number of records, i.e., lines (ADVANCE)
 - c) Forward or backward a specified number of characters (JUMP)
 - d) To a specified group of characters (FIND, GET, POSITION)
2. Modification of file records
 - a) By removing characters or records (DELETE, KILL)

- b) By adding characters or records (INSERT)
 - c) By altering lines (CHANGE)
 - d) By replacing lines (EXCHANGE)
3. Manipulation of editor buffers and files (Note: This editor defines its process in terms of the input file, the output file, and four logical buffers: Text, Command Input, Macro and Save. A single page of about 55 records from the input file may occupy the text buffer at a given time. Text and pointer modification commands usually apply only to data in Text buffer.)
- a) By opening a file for input (EDIT READ, EDIT BACKUP)
 - b) By preparing a file for output of text (EDIT WRITE, EDIT BACKUP)
 - c) By moving a page from the input file to the Text Buffer (READ, NEXT)
 - d) By moving records from the Text Buffer to the output file (WRITE, NEXT)
4. Definition and manipulation of command and text groupings
- a) By allowing lines of text to be temporarily stored for later insertion into files (SAVE, UNSAVE)
 - b) By allowing strings of edit commands to be stored and later executed repetitively (MACRO, EXECUTE MACRO)
5. Display of text and editor information
- a) By printing all specified records (LIST)
 - b) By printing the current line, i.e., the one containing the pointer (VERIFY)
 - c) By printing the version number of the editor (EDIT VERSION)

6. Maintenance of editor files

- a) By storing the output file and exiting the editor (EXIT)
- b) By storing the output file without exiting the editor (END FILE)
- c) By exiting the editor without storing any files (C)

7. Modification of textual display

- a) By allowing both upper and lower case display or only upper-case display (EDIT LOWER, EDIT UPPER)
- b) By allowing screen display of part of text Buffer commands or of commands only (EDIT CONSOLE, EDIT DISPLAY)

II. Functional capabilities of Type II editors

1. Modification of pointer position.

- a) To last line of file (BOTTOM)
- b) To top of file (TOP)
- c) Toward bottom of file (DOWN, NEXT)
- d) Toward top of file (UP)
- e) To a specified group of characters (FIND, LOCATE)
- f) To a specified line number (GO TO)

2. Modification of file records

- a) By removing lines (DELETE)
- b) By adding lines (INPUT, INSERT)
- c) By altering lines (CHANGE)
- d) By replacing lines (REPLACE)
- e) By overlaying lines with blanks or with specified lines (BLANK, OVERLAY)

- f) By fetching or placing designated records from/
to another file or temporary buffer (GET, PUT, PUTD)

3. Modification of EDIT command modes

- a) By displaying file contents in hexadecimal and interpreting command text strings as hexadecimal (HEX)
- b) By suppressing automatic verification and other system responses (BRIEF, SBRIEF)
- c) By allowing files to be edited with record numbers (ENTER, RENUMBER, SERIAL)
- d) By limiting columns scanned in commands (ZONE)
- e) By building command macros (X, Y, XX, and YY Macros)

4. Definition and modification of character set

- a) By defining logical backspace (BACKSPACE)
- b) By defining and setting logical tabs (TABDEF, TABSET)
- c) By defining break character for edit macros (SETBREAK)
- d) By defining equivalence for non-printing character (SETCHAR)

5. Display of lines and line numbers (PRINT, VERIFY, LINE NO)

6. Automatic repetition of commands (AGAIN)

7. Maintenance of editor files

- a) By erasing temporary buffers (ERASE)
- b) By storing the file as edited and exiting editor (FILE)
- c) By exiting editor without storing the file as edited (QUIT)
- d) By storing files without exiting editor (SAVE, AUTO SAVE)

8. Clarification of Editor command errors (?)
9. Communication with system monitor while remaining in EDIT mode (CSS)

III. Functional Capabilities of a Type III Editor

1. Modification of record numbers (RESEQ). Note that line number references are used instead of a pointer.
2. Modification of file records
 - a) By removing lines (DELETE)
 - b) By adding lines (ADD, LINE-TEXT)
 - c) By replacing lines (TEXT REPLACEMENT)
3. Definition and modification of file format (FORMAT)
4. Storage and manipulation of files
 - a) By creating a new file (CREATE)
 - b) By editing an already existing file (EDIT)
 - c) By saving an edited file without exiting editor (SAVE)
5. Display of specified lines (LIST)
6. Compilation and execution of programs (RUN)
7. Exit from editor (BYE, END)

Requirement 3.1.1.d.1

Classification: Basic

Requirement:

Add one or more characters to a file without re-entering the entire record.

Purpose of Paragraph:

To specify the smallest unit of data which may be inserted into a file, and to define limitations upon such insertion.

Existing Tools which Satisfy this Requirement:

The requirement may be satisfied by an editor or utility which includes a command which allows insertion of a character or characters into a user-indicated record.

Character insertion is usually accomplished by a CHANGE command which has as parameters:

- (1) Place of insertion indicated by unique character(s) which immediately follow or precede the place of insertion;
- (2) The unique character(s) preceded (or followed) by the character(s) to be inserted.

Parameters are separated by delimiting characters which, in most systems, may be any character not contained within the given parameters themselves.

Examples:

Example 3.1.1.d.1.1: Addition of one or more characters to a file using a Type I system.

Once the editor is invoked and the user receives the editor prompt of an asterisk, a copy of the desired file, SAMPLE.DAT, is accessed by the EB (edit backup) command. Note that a dollar sign (character representing the "alt mode" key on this system's terminal) is the command delimiter, while two delimiters mark the end of a series of commands.

This system requires that the user expressly read the file into the editor's buffer, which is accomplished by the R command. With this done, the user has available to him the commands for editing which were discussed in the Existing Tools section of paragraph 3.1.1.d. In the sample dialogue, the user advances to the third record in the file and requests the line be verified, i.e., displayed, by typing "3A\$V\$\$". Note that this system uses a typical pointer technique to indicate the record to be altered.

Addition of characters may be done in two ways. If the new characters are to appear at the beginning of the record, then the I command is given, followed immediately by the string to be added. If the new characters appear mid-record, the user selects characters which uniquely identify the point of insertion, and changes that point in the record to include the new string. In the given sample, the first 5. is located with the G command, then changed with the C command to include "4.4.44". Change verification is accomplished by moving to the beginning of the file and listing the file in its entirety (B/L\$\$). Upon exiting from the editor the altered copy of the file is placed after the old copy stored under the same name.

While this system is not as flexible as that discussed in the next example, it does allow the user to perform all major editing functions.

```
. R E
#ERSAMPLE.DAT$$
* R $$
*3A$V$$
  2.221 37.637 5.838 96.291 25.208
*I 11.111$V$$
  11.111 2.221 37.637 5.838 96.291 25.208
*G5.9=C4.444 5.$V$$
  11.111 2.221 37.637 4.444 5.838 96.291 25.208
*B/L$$
  6.052 0.412 1.730 6.674 24.472
  86.765 0.341 21.157 23.878 52.854
  11.111 2.221 37.637 4.444 5.838 96.291 25.208
  84.626 80.885 23.679 14.110 71.548
```

Figure 3.1.1.d.1-1. Adding one or more units of data to a file via a Type I system editor.

Example 3.1.1.d.1.2: Addition of one or more characters to a file using a Type II System.

Most commercial time share systems allow greater flexibility in editing than do small computer systems, as is shown in Figure 3.1.1.d.1-2 which parallels that of the preceding example. Note that invocation of the editor, designation of the file to be edited, and reading of the file into the editor's buffer are all done in one command. The system responds with a statement of the new operational mode ("EDIT:"), but the system prompt is not altered. When the pointer is advanced by the N command (abbreviation for NEXT), the record pointed to is displayed without additional commands.

One command the system supports for character insertion into a record is the CHANGE command. The location of the change and the new string are given as two delimited parameters following the command. Insertion at the beginning of a record is done by giving a null string as the first parameter.

In the example, the pointer is moved to the top of the file and the file listed by "T#P99". Following verification, the FILE command causes the altered file to be stored under the same name, though it may be stored separately by giving a new file name as parameter in the file command. The editor is exited at the same time.

The editor commands in a commercial time share system are typically more powerful and broader in scope, allowing the user to issue a single command to achieve the same result as is accomplished by several commands in a smaller system.

```

08.34.14 >EDIT SAMPLE DATA
EDIT:
>N 3
      2.221  37.637   5.838  96.291  25.208
>C // 11.200/
      11.200   2.221  37.637   5.838  96.291  25.208
>C /5./7.777  5./
      11.200   2.221  37.637   7.777   5.838  96.291  25.208
>P 99
      0.082   0.412   1.730   6.674  24.472
      86.765   0.341  21.157  23.878  52.854
      11.200   2.221  37.637   7.777   5.838  96.291  25.208
      84.626  80.885  23.679  14.110  71.548
>FILE

08.36.42 >

```

Figure 3.1.1.d.1-2. Adding one or more units of data to a file via a Type II system editor.

Example 3.1.1.d.1.3: Addition of one or more characters to a file using a Type III system.

The limited interactive facilities of a Type III system put it at a disadvantage in file editing. In the sample dialogue below, it is first necessary to establish a link between the permanent storage disk and the user, and to assign a working file name to the file, as well as to give the name under which it is stored. This is done in the ATTACH command. The working file name could have been omitted, in which case the system would have assigned a name. The editor is invoked by the command EDITOR, and the system responds with a double period, the editor prompt. It is not necessary to specifically state that the work file is to be edited and sequence numbers assigned to the records (E,SMP,S). This system assigns sequence numbers of 100 for the first record, 110 for the second, etc. Therefore, in order to access the third line, the "L,120" command is given. Insertion into a record is accomplished by giving as parameters the location of the change in terms of unique characters, and the full replacement string. These parameters must be delimited by slashes (unless otherwise specified) and separated by an equal sign. Note that the first parameter may not be null and must be unique, every occurrence of the first will be replaced by the second parameter.

Change verification may be done in two ways. In the example, a separate LIST command is issued, with the specific record number to be listed. The same purpose may have been accomplished by appending a verify request to the change parameters; however, this would have required the user to respond Y for yes or N for no to invoke the change.

Insertions in the middle of a line are done in the same manner as those to the beginning of a line. Finally, the entire file may be listed by specifying A for ALL in the LIST command.

Storage of the altered file is considerably more complex in this system than in the other two. The new file version must be saved under a local name different from the work file name. In the example, the command "S,X,N" does this, with X being the local file name and S the save command. The N indicates that the sequence numbers are not to be stored with the file. The editor may now be exited via the BYE command. The file must be permanently saved, as the new version, X, is only temporarily saved. Thus, the CATALOG command is issued, followed by the local file name, the permanent file name, and the user identification.

This system is substantially more awkward than the others presented. However, despite its drawbacks, it does fulfill the requirements demanded of the TDSP standard.

COMMAND- ATTACH, SMP, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 001
COMMAND- EDITOR

..E, SMP, S
..L, 120

120= 1.221 37.637 5.838 96.291 25.208

../ 1./= /11.300 1./
I CHANGES

..L, 120

120= 11.300 1.221 37.637 5.838 96.291 25.208

../5.8/= /3.333 5.8/
I CHANGES

..L, 120

120= 11.300 1.221 37.637 3.333 5.838 96.291 25.208

..L, A

100= 0.082 0.412 1.730 6.674 24.472
110= 86.765 0.341 21.157 23.878 52.854
120= 11.300 1.221 37.637 3.333 5.838 96.291 25.208
130= 84.626 80.885 23.679 14.110 71.548

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEDATA, ID=GAERTNER

NEW CYCLE CATALOG
RP = 090 DAYS
CT ID= GAERTNER PFN= SAMPLEDATA
CT CY= 002 0000128 WORDS.:
COMMAND-

Figure 3.1.1.d.1-3. Adding one or more units of data to a file using a Type III system.

Requirement 3.1.1.d.2

Classification: Basic

Requirement:

Delete one or more characters from a file without re-entering the entire record containing those characters.

Purpose of Paragraph:

To specify the smallest unit of data which may be deleted from a file, and to define limitations upon such deletion.

Existing Tools which Satisfy this Requirement:

This requirement may be satisfied by an editor or utility which includes a command allowing deletion of a character or characters in a user-indicated record.

Examples:

Example 3.1.1.d.2.1: Deletion of one or more characters from a file using a Type I system.

Character deletion in this sample system is accomplished using the Get and Change commands together, much in the same manner as Figure 3.1.1.d.1-1, to define location within record of deletion and request the change. Deletion is accomplished by specifying a null string as replacement for the characters to be deleted.

Looking at the sample dialogue below, once again the editor is invoked, the file accessed, and made available to the editor. The file pointer is moved to the desired record, and the beginning eight characters described in the Get command. These are then changed to a null string. A similar procedure is followed to eliminate mid-record characters in the next command string.

As in the last example on this system, verification of the alteration is done by moving the pointer to the top of the file and listing the entire file. As before, the new version is stored, replacing the old, upon exit from the editor.

COMMAND- ATTACH, SMP, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 001
COMMAND- EDITOR

..E, SMP, S
..L, 120

120= 1.221 37.637 5.838 96.291 25.208

../ 1./=/11.300 1./
[CHANGES

..L, 120

120= 11.300 1.221 37.637 5.838 96.291 25.208

../5.8/=/3.333 5.8/
[CHANGES

..L, 120

120= 11.300 1.221 37.637 3.333 5.838 96.291 25.208

..L, A

100= 0.082 0.412 1.730 6.674 24.472

110= 86.765 0.341 21.157 23.878 52.854

120= 11.300 1.221 37.637 3.333 5.838 96.291 25.208

130= 84.626 80.885 23.679 14.110 71.548

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEDATA, ID=GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN=SAMPLEDATA

CT CY= 002 0000128 WORDS.:

COMMAND-

Figure 3.1.1.d.1-3. Adding one or more units of data to a file using a Type III system.

Requirement 3.1.1.d.2

Classification: Basic

Requirement:

Delete one or more characters from a file without re-entering the entire record containing those characters.

Purpose of Paragraph:

To specify the smallest unit of data which may be deleted from a file, and to define limitations upon such deletion.

Existing Tools which Satisfy this Requirement:

This requirement may be satisfied by an editor or utility which includes a command allowing deletion of a character or characters in a user-indicated record.

Examples:

Example 3.1.1.d.2.1: Deletion of one or more characters from a file using a Type I system.

Character deletion in this sample system is accomplished using the Get and Change commands together, much in the same manner as Figure 3.1.1.d.1-1, to define location within record of deletion and request the change. Deletion is accomplished by specifying a null string as replacement for the characters to be deleted.

Looking at the sample dialogue below, once again the editor is invoked, the file accessed, and made available to the editor. The file pointer is moved to the desired record, and the beginning eight characters described in the Get command. These are then changed to a null string. A similar procedure is followed to eliminate mid-record characters in the next command string.

As in the last example on this system, verification of the alteration is done by moving the pointer to the top of the file and listing the entire file. As before, the new version is stored, replacing the old, upon exit from the editor.

It is interesting to compare the editors of the small computer system and the commercial time-sharing system. The commands of the Type II editor are "high level language" whereas the commands of the Type I editor are "assembly level language". For this reason, the Type II editor supports a better user interface, and is, in almost every way, better than the Type I editor.

The Type III system, on the other hand, is marked by its interactive limitations, and falls far short of the other two in terms of ease of usage. While character addition is not much more difficult, file access and restoration are much more awkward, requiring two temporary names which are different from the permanent name.

```
.R E
#E SAMPLE.DATSR53ASVSS
  11.111  2.221  37.637  4.444  5.838  96.291  25.208
*G 11.111S=CSVSS
  2.221  37.637  4.444  5.838  96.291  25.208
*G4.444  S=CSVSS
  2.221  37.637  5.838  96.291  25.208
*BLSS
  0.002  0.412  1.730  6.674  24.472
  86.765  0.341  21.157  23.878  52.854
  2.221  37.637  5.838  96.291  25.208
  84.626  80.885  23.679  14.110  71.548
*EXSS
```

Figure 3.1.1.d.2-1. Deleting one or more characters from a file using a Type I system.

Example 3.1.1.d.2.2: Deletion of one or more characters from a file using a Type II system.

In the commercial time sharing system, the commands for deletion are identical to those for data addition; only the parameter values are different. Once the record to be changed has been located via the NEXT command, the CHANGE command is issued with the first parameter being the character(s) to be deleted, and the second parameter a null string. Verification is accomplished, as before, by moving to the top of the file (TOP command) and displaying it with the PRINT command.

```
10.02.13 >EDIT SAMPLE DATA
EDIT:
>N 3
    11.200    2.221    37.637    7.777    5.838    96.291    25.208
>C / 11.200//
    2.221    37.637    7.777    5.838    96.291    25.208
>C / 7.777//
    2.221    37.637    5.838    96.291    25.208
>TOP 9
    0.002    0.412    1.730    6.674    24.472
    86.765    0.341    21.157    23.878    52.854
    2.221    37.637    5.838    96.291    25.208
    84.626    80.885    83.679    14.110    71.548
>FILE

10.04.56 >
```

Figure 3.1.1.d.2-2. Deleting one or more characters from a file using a Type II system.

Example 3.1.1.d.2.3: Deletion of one or more characters from a file using a Type III System. Sample System: CDC 6600, Scope.

As in the other two systems, deletion of characters utilizes a method similar to character insertion. In the dialogue below, the file is accessed as before, and the third line is made the current line by listing it. The characters to be deleted are then given as the first parameter, while a null string is given as second parameter in the change expression. Once again, verification is accomplished by the list command, after which the file is restored in the same manner as in Figure 3.1.1.d.1-3.

Requirement 3.1.1.d.3

Classification: Basic

Requirement:

Replace one or more characters from a file without re-entering the entire record.

Purpose of Paragraph:

To specify the smallest unit of data which may be replaced in a file, and to define limitations upon such replacement.

Existing Tools which Satisfy this Requirement:

This requirement may be satisfied by any editor or utility which includes a command allowing replacement of one string of characters with another string of perhaps differing length.

Examples:

Example 3.1.1.d.3.1: Replacing one or more characters with a new character or characters using a Type I system.

The same set of commands are used for replacement as were used for addition and deletion. As may be seen in the sample dialogue below, the characters to be replaced are indicated with the GET command, while the replacement characters are given in the change command.

COMMAND- ATTACH, SMP, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 002

COMMAND- EDITOR

..E, SMP, S

..L, 120

120= 11.300 1.221 37.637 3.333 5.838 96.291 25.208

../11.300 /=//

..I CHANGES

..L, 120

120= 1.221 37.637 3.333 5.838 96.291 25.208

../3.333 /=//

..I CHANGES

..L, 120

120= 1.221 37.637 5.838 96.291 25.208

..L, A

100= 0.032 0.412 1.730 6.674 24.472

110= 86.765 0.341 21.157 23.878 52.854

120= 1.221 37.637 5.838 96.291 25.208

130= 84.626 80.885 23.679 14.110 71.548

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEDATA, ID=GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN=SAMPLEDATA

CT CY= 003 0000128 WORDS.:

COMMAND-

Figure 3.1.1.d.2-3. Deleting one or more characters from a file using a Type III system.

Example 3.1.1.d.3.2: Replacing one or more characters with a new character or characters using Type II and Type III systems.

Both these systems accomplish additions and deletions with a two parameter CHANGE command. Replacement of characters is likewise done the same way. In each of the 2 sample dialogues below, parameter one is the string of characters to be replaced, while parameter two is the replacement string. The remainder of each example follows the same procedure as was illustrated before in examples for the corresponding system.

```
.R E
#BSAMPLE.DATSR$JASV$$
  2.221 37.637 5.838 96.291 25.208
#637.637$=C33.345$V$$
  2.221 33.345 5.838 96.291 25.208
#096.2$=C22.9$V$$
  2.221 33.345 5.838 22.991 25.208
#B/L$$

  0.852 0.412 1.730 6.674 24.472
  86.765 0.341 21.157 23.878 52.854
  2.221 33.345 5.838 22.991 25.208
  84.626 80.885 23.679 14.110 71.548
#EX$$
```

Figure 3.1.1.d.3-1. Replacing one or more characters using a Type I system.

```

101.46.23 >EDIT SAMPLE DATA
EIT:
>N 3
    2.221  37.637  5.838  96.291  25.288
>C /37.637/22.234/
    2.221  22.234  5.838  96.291  25.288
>C /25.288/55.59/
    2.221  22.234  5.838  96.291  55.598
*P 9
    0.882  0.412  1.730  6.674  24.472
    86.765  0.341  21.157  23.878  52.854
    2.221  22.234  5.838  96.291  55.598
    84.626  80.885  23.679  14.110  71.548
>FILE

101.47.38 >

```

Figure 3.1.1.d.3-2. Replacing one or more characters using a Type II system editor.

COMMAND- ATTACH, SMP, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 003
COMMAND- EDITOR

..E, SMP, S
..L, 120

120= 1.221 37.637 5.838 96.291 25.208

../1.22/= /3.99/
I CHANGES

..L, 120

120= 3.991 37.637 5.838 96.291 25.208

../.291/= /.629/
I CHANGES

..L, 120

120= 3.991 37.637 5.838 96.629 25.208

..L, A

100=	0.082	0.412	1.730	6.674	24.472
110=	86.765	0.341	21.157	23.878	52.854
120=	3.991	37.637	5.838	96.629	25.208
130=	84.626	80.885	23.679	14.110	71.548

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEDATA, ID=GAERTNER

NEW CYCLE CATALOG
RP = 090 DAYS
CT ID= GAERTNER PFN=SAMPLEDATA
CT CY= 004 0000128 WORDS.:
COMMAND-

Figure 3.1.1.d.3-3. Replacing one or more units of data with new units of data using a Type III system.

Requirement 3.1.1.d.4

Classification: Basic

Requirement:

Make the following changes to one or more records of source data:

- (a) Add one or more records of data.
- (b) Delete one or more records of data.
- (c) Replace one or more records of data.

Purpose of Paragraph:

To define the types of operations upon a record necessary to implement a PSL.

Existing Tools Which Satisfy this Requirement:

This requirement may be satisfied by any editor which contains record (line) oriented commands for addition, deletion, and replacement.

Examples:

Whereas the preceding requirements concerned the alteration of characters within a record, this requirement defines the necessary operations upon a record within a file. In this context, the term "line" will be used interchangeably with "record", though, remember that the line referred to here is not that defined in SPS V-3.1.1.d.4.a, but a more general one, the length of which is dependent upon the display media and type of file being edited.

- (a) Addition of a record to a file.

Example 3.1.1.d.4.a.1: Addition of one or more records to a file using a Type I system.

Again using the minicomputer editing system shown in preceding examples, the editor is first invoked and the file accessed. Once a page of the file has been read into the editor buffer (R command), the actual editing begins.

First, the pointer is positioned by moving it past N end-of-line indicators to the beginning of the N+1'th line. An insert command is issued and the new line is typed. All data typed thereafter will be considered to be input until the user exits input mode by issuing some modal delimiter, in this case the "\$" (i.e., the Alt Mode key symbol). The new material has been placed in the file between the initial pointer position and the beginning of the N+1'th line.

Our sample dialogue below shows one line inserted prior to line three, then two lines added before old line four.

As before, the entire file is displayed by moving to the beginning and issuing the "/L" command, then the new version of the file is stored simply by exiting from the editor.

```

• R E
* EBSAMPLE. FOR $R$3A$V$$
      BESTFT=(A+A2)/2.
* I      SAVSUM=A+A2
$-L $$      SAVSUM=A+A2
* A$ I      HAUG=SAVSUM +TRANS +REGIN
      TGRET=HAUG /PI
$B/L $$
      A=C(I)
      A2 =C(I+1)
      SAVSUM=A+A2
      BESTFT=(A+A2)/2.
      HAUG=SAVSUM +TRANS +REGIN
      TGRET=HAUG /PI
      GAUSIN=FGAUS( BESTFT )
* EX $$

```

Figure 3.1.1.d.4.a-1. Adding one or more records to a file using a Type I system editor.

Example 3.1.1.d.4.a.2: Addition of one or more records to a file using a Type II system.

CSS, offers two commands for line addition: INSERT, followed by a blank and a character string, to insert a single line; INPUT to change into an input mode. In either case the file pointer is positioned at the record after which the new material is to appear. This pointer is advanced to the new line after it has been added.

In the sample dialogue below, the editor is entered, the file accessed, and the pointer moved to line two, which is then displayed. A single line is inserted by entering I, blank, and the new line. The pointer is then advanced one more record to the old fourth (new fifth) line, and input mode entered. The system confirms the new mode and all typed material is inserted into the file until a null line is entered. As before, the entire file is displayed and then saved via the FILE command.

EXAMPL23 MEMO P ----- 20APR79 ----- PAGE 1

```
22.08.50 >edit sample fortran
EDIT:
>next 2
      A2=C(I+1)
>insert      savsum=a+a2
>print
      SAVSUM=A+A2
>next
      BESTFT=(A+A2)/2.
>input
INPUT:
>      havg=savsum+trans+resin
>      tgret=havg/pi
>
EDIT:
>top
>print 99
      A=C(I)
      A2=C(I+1)
      SAVSUM=A+A2
      BESTFT=(A+A2)/2.
      HAVG=SAVSUM+TRANS+REGIN
      TGRET=HAVG/PI
      GAUSIN=FGAUS(BESTFT)

EOF:
>file

22.11.58 >
```

Figure 3.1.1.4.a-2. Addition of one or more records to a file using a Type II system.

Example 3.1.1.d.4.a.3: Addition of one or more records to a file using a Type III system.

Like the Type II system, the CDC interactive editor supports two methods of line insertion: One for single line entries and one for multiple line entries. Both, however, are tied to the line sequence number, necessitating that the line to be inserted have integer numbers which indicate the desired line sequence. If the proposed addition will cause an overlap in line numbers with an already existing line, then the entire file must have its lines resequenced.

After the initial editor linkage and file access, the sample dialogue below shows the insertion of a single line, a new third line, by assigning it a number which is between the current sequence numbers of the second and third lines. Addition of multiple lines requires entering the add mode via the ADD command. This takes two parameters: The number of the starting line and the increment for successive line numbers (A, 121, 1). The system response is the line number and an "=". The user then enters the desired line. Exit from the add mode is done by entering a single equal sign. File verification and storage are as shown in preceding Type III system examples.

COMMAND- ATTACH, F, SAMPLEFORT, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- EDITOR

..E, F, S

..115= SAVSUM=A+A2

A,121,1

121= HAUG=SAVSUM +TRANS +REGIN

122= TGRET=HAUG /PI

123==

..L, A

100= A=C(I)

110= A2 =C(I+1)

115= SAVSUM=A+A2

120= BESTFT=(A+A2)/2.

121= HAUG=SAVSUM +TRANS +REGIN

122= TGRET=HAUG /PI

130= GAUSIN=FGAUS(BESTFT)

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEFORT, ID=GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN=SAMPLEFORT

CT CY= 002 0000128 WORDS.:

COMMAND-

Figure 3.1.1.d.4.a-3. Adding one or more records to an existing file using a Type III system.

General Discussion on Record Insertion:

Insertion of records requires that the user be aware of file size limitations of the system utilized. Most small computer systems have a moderately low default limit. For example, the PDP-11 system used to generate Type I system examples normally accepts files of less than 500 records, though this size may be increased by special options. The same case holds with the time sharing system, though the limit there is considerably larger (one million characters), and there is a maximum limit of 168 million bytes. The sample Type III system has no fixed file limitation; rather, size boundaries are defined for individual installations.

Within a given file, Types I and II systems have no limit (other than stated above) on the amount of material that may be added at a given time. The Type III system, however, will not allow a new line number to be equal to or greater than an already existing line number unless a special option is exercised. Should an attempt be made to insert a line with an improper number, the system responds with an error message. If the default numbering system with an increment of 10 is used, then only nine lines may be inserted. Thus, the re-sequencing feature is required, especially when comments are to be inserted into source code. This characteristic hampers program documentation and therefore makes the Type III system inferior for implementation of a PSL.

(b) Deleting Source Records

Example 3.1.1.d.4.b.1: Deletion of one or more records from a file using a Type I system.

Line deletion is accomplished by positioning the pointer at the record (or first record) to be deleted and issuing the KILL command. If no parameters accompany, only the line pointed to is deleted. A number preceding the command, however, indicates the number of lines (including the current line) which are to be deleted.

In the sample dialogue below, the lines that were added in the previous example are now deleted. Note that commands have been strung together to facilitate editing.

```

• R E
*EBSAMPLE.FORSR$3ASVSS
    SAVSUM=A+A2
*KSASVSS
    HAUG=SAVSUM +TRANS +REGIN
*2KSVSS
    GAUSIN=FGAUS( BESTFT )
*B/LSS
    A=C(I)
    A2 =C(I+1)
    BESTFT=(A+A2)/2.
    GAUSIN=FGAUS( BESTFT )
*EXSS

```

Figure 3.1.1.d.4.b-1. Deleting one or more records using a Type I system.

Example 3.1.1.d.4.b.2: Deletion of one or more records from a file using a Type II system.

Line deletion in the sample time-sharing system is similar to that of the Type I system. The pointer is positioned either with the D (DOWN) or N (NEXT) command, then deleted with the DE (DELETE) command.

```

11.49.07 >EDIT SAMPLE FORTRAN
EDIT:
>D 3
    SAVSUM=A+A2
>DE
>N
    HAUG=SAVSUM +TRANS +REGIN
>DE 2
>P
    GAUSIN=FGAUS( BESTFT )
>TOP 9
    A=C(I)
    A2 =C(I+1)
    BESTFT=(A+A2)/2.
    GAUSIN=FGAUS( BESTFT )
>FILE

11.51.02 >

```

Figure 3.1.1.d.4.b-2. Deletion of one or more records from a file using a Type II system.

Example 3.1.1.d.4.b.3: Deletion of one or more records from a file using a Type III system.

Since the sample Type III system uses the line number rather than a pointer to position with a file, deletion of records will require a sequence number as parameter following the command (DELETE or D). If more than one line is to be deleted, the number of the first deletion line is parameter one and the sequence number of the last record to be deleted is the second parameter. It should be noted in the sample below that the sequence numbers assigned to records are different from the end of the last editing session (Example 3.1.1.d.4.a.3). When the editor reassigned sequence numbers (E, F, S), the default sequence was assigned. A new listing of the entire file is required in order to delete accurately.

(c) Replacement of Source Records

Example 3.1.1.d.4.c.1: Replacement of one or more records within a file using a Type I system.

Record replacement in this typical small system is done in two ways: Using the X (EXCHANGE) instruction for one-to-one line replacement; using a combination of KILL (line delete function) and INSERT for other cases.

In the sample dialogue below, the editor is invoked and file accessed as usual. After listing the file and positioning the pointer at the beginning, the X instruction causes the character string parameter to replace the record currently being pointed to (the first record). This command also advances the pointer one line, so the verify request then causes line two to be printed. Next lines 2 and 3 are deleted and two other lines inserted in their place (the X instruction could have been used here).

COMMAND- ATTACH, F, SAMPLEFORT, ID=GAERTNER

PF CYCLE NO.= 002
COMMAND- EDITOR
..E,F,S

..L,A

100=	A=C(I)
110=	A2 =C(I+1)
120=	SAVSUM=A+A2
130=	BESTFT=(A+A2)/2.
140=	HAVG=SAVSUM +TRANS +REGIN
150=	TGRET=HAVG /PI
160=	GAUSIN=FGAUS(BESTFT)

..D,120

..D,140,150

..S,X,N

..BYE

COMMAND- CATALOG,X, SAMPLEFORT, ID=GAERTNER

NEW CYCLE CATALOG
RP = 090 DAYS
CT ID= GAERTNER PFN=SAMPLEFORT
CT CY= 003 0000128 WORDS.:
COMMAND-

Figure 3.1.1.d.4.b-3. Deleting one or more records using a Type III system.

Example 3.1.1.d.4.b.3: Deletion of one or more records from a file using a Type III system.

Since the sample Type III system uses the line number rather than a pointer to position within a file, deletion of records will require a sequence number as parameter following the command (DELETE or D). If more than one line is to be deleted, the number of the first deletion line is parameter one and the sequence number of the last record to be deleted is the second parameter. It should be noted in the sample below that the sequence numbers assigned to records are different from the end of the last editing session (Example 3.1.1.d.4.a.3). When the editor reassigned sequence numbers (E, F, S), the default sequence was assigned. A new listing of the entire file is required in order to delete accurately.

(c) Replacement of Source Records

Example 3.1.1.d.4.c.1: Replacement of one or more records within a file using a Type I system.

Record replacement in this typical small system is done in two ways: Using the X (EXCHANGE) instruction for one-to-one line replacement; using a combination of KILL (line delete function) and INSERT for other cases.

In the sample dialogue below, the editor is invoked and file accessed as usual. After listing the file and positioning the pointer at the beginning, the X instruction causes the character string parameter to replace the record currently being pointed to (the first record). This command also advances the pointer one line, so the verify request then causes line two to be printed. Next lines 2 and 3 are deleted and two other lines inserted in their place (the X instruction could have been used here).

```

•R E
•EBSAMPLE.DAT$RSDS/LSS
    A=C(I)
    A2 =C(I+1)
    BESTFT=(A+A2)/2.
    GAUSIN=FGAUS( BESTFT )
•X    BASEA=CDEC +BSUMA
SVSS
    A2 =C(I+1)
•2K M    BASEAA =( WSTRL + ESTRL + YQAST )/ PI
    BESTFT =BESTFT + BASEA +2.*BASEAA
SVSS
    GAUSIN=FGAUS( BESTFT )
•B/LSS
    BASEA=CDEC +BSUMA
    BASEAA =( WSTRL + ESTRL + YQAST )/ PI
    BESTFT =BESTFT +BASEA +2.*BASEAA
    GAUSIN=FGAUS( BESTFT )
•EXSS

```

Figure 3.1.1.d.4.c-1. Replacing one or more records via a typical Type I system.

Example 3.1.1.d.4.c.2: Replacement of one or more records within a file using a Type II system.

The commercial time-sharing system offers a typically more flexible version of the replacement command. Here the REPLACE command allows the user to replace one line with one or more lines, as desired. Any remaining lines which are not wanted may then be deleted.

In the sample dialogue below, a signal line is replaced in the same manner as in the Type I system. If more than one line is to be inserted, the REPLACE command is issued without parameters. This places the user in input mode, and he may then type in all desired material. Exit from input mode, as before, is achieved by entering a null line. The REPLACE command always leaves the pointer positioned at the last new line entered.

```
22.18.06 >e sample fortran
EDIT:
>print 9
      A=C(I)
      A2=C(I+1)
      BESTFT=(A+A2)/2.
      GAUSIN=FGAUS(BESTFT)
EOF:
>top
>next
      A=C(I)
>replace      base=ccss+codval
>down
      A2=C(I+1)
>r
INPUT:
>      baseaa=(wstrl+estrl+tqast)/pi
>      bestft=bestft+basec+2.*baseaa
>
EDIT:
>p
      BESTFT=BESTFT+BASEC+2.*BASEAA
>d
      BESTFT=(A+A2)/2.
>delete
>top
>p 9
      BASC=CCSS+CODVAL
      BASEAA=(WSTRL+ESTRL+TQAST)/PI
      BESTFT=BESTFT+BASEC+2.*BASEAA
      GAUSIN=FGAUS(BESTFT)
EOF:
>file

22.21.51 >
```

Figure 3.1.1.d.4.c-2. Replacement of one or more records within a file using a Type II system.

Example 3.1.1.d.4.c.3: Replacement of one or more records within a file using a Type III system.

Replacement of one line of source code in this sequence number oriented system simply requires that the sequence number of the line be typed, followed by an equal sign and the replacement line. Multiple line replacement may be done in two ways. The first and least error-prone method is to delete the unwanted records and then add the new material. This is the method used in the example below.

The other way to replace records is to issue the ADD command with the parameters adjusted accordingly and the OVERWRITE option set (e.g., A, 110, 10, 0). This allows the user to overwrite lines which have already existing sequence numbers. However, since those lines are not displayed before overwriting, it is very easy to overwrite valid material, thus making this method unsafe.

Requirement 3.1.1.d.5

Classification: Basic

Requirement:

Organize data for easy reference by providing:

- (a) Automatic generation of record sequence numbers within files for data added to the library.
- (b) Regeneration of record sequence numbers as a result of changes to data.
- (c) The capability to add records to an existing file without regeneration of record sequence numbers.

Purpose of Paragraph:

To define the capabilities required for reference of records within a file.

Existing Tools which Satisfy the Requirement:

This requirement may be satisfied by any editor which supplies record sequence numbers but which also allows a file to exist without sequencing.

COMMAND- ATTACH, F, SAMPLEFORT, ID=GAERTNER

PF CYCLE NO.= 003

COMMAND- EDITOR

..E, F, S

..L, A

```
100=      A=C(I)
110=      A2 =C(I+1)
120=      BESTFT=(A+A2)/2.
130=      GAUSIN=FGAUS( BESTFT )
```

..100= BASEC=CCDC +CODVAL

..D, 110, 120

..A, 110, 10

110= BASEAA=(WSTRLY +ESTRLY +YQAST) / PI

120= BSETFT=BESTFT +BASEC +2.*BASEAA
ADD WONT REPLACE OR BYPASS LINES

..L, A

```
100=      BASEC=CCDC +CODVAL
110=      BASEAA=(WSTRLY +ESTRLY +YQAST) / PI
120=      BESTFT=BESTFT +BASEC +2.*BASEAA
130=      GAUSIN=FGAUS( BESTFT )
```

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEFORT, ID=GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN=SAMPLEFORT

CT CY= 004 0000128 WORDS.:

COMMAND-

Figure 3.1.1.d.4.c-3. Replacing one or more records using a Type III system.

Examples:

Once again the terms "line" and "record" shall be used interchangeably in the discussion to follow. The definitions remain as stated at the beginning of the examples for SPS V 3.1.1.d.4.

(a) Automatic generation of record sequence numbers.

This feature is not available in the DEC RT-11 editor which has been used to generate most of the Type I examples. However, other minicomputer editors do exist which allow this feature.

In the case where this feature does not already exist, the prospective Government contractor has two options: To modify the existing editor or to obtain additional software which would supply record sequence numbers. The first method not only causes the prospective contractor to incur additional programming expense, it also creates an editor which is unique to one installation. For that reason the second option is considered superior. Such additional software may be supplied by the Government as an element of the library for structured programming support. The specifications for this program are detailed in the Paragraph entitled "Program Requirements for Necessary New Support Software".

Example 3.1.1.d.5.a.1: Automatic generation of record sequence numbers using a Type II system.

The time-sharing systems encountered all provided facilities to meet this requirement. They also provide commands that allow the user to alter the starting point of the sequence numbering as well as the increment value. The CMS system provides line numbers automatically only for those file types which have a fixed record length of 80 characters. In these the sequence number appears in columns 76-80 and a three letter identifier in the preceding three columns. The same method of sequence numbering is used by CSS, but in that system the user must disable serialization for files that are to contain data in columns 73 through 80. Figures 3.1.1.d.5.a-1 and 3.1.1.d.5 a-2 below provide illustrations of files with sequence numbers, the first being a data file and the second a source file.

Example 3.1.1.d.5.a.2: Automatic generation of record sequence numbers using a Type III system.

As has been mentioned before, the CDC Intercom is line sequence number oriented, and therefore satisfies this requirement by the initial system design. These numbers are automatically stored with the file unless otherwise specified via the N option of the SAVE command. However, since the sequence numbers are moved to the end of the line prior to storage, and intervening blanks are not compressed, retention of sequence numbers with a file wastes storage.

Sample dialogues which illustrate the Type III line numbering system may be found in Figures 3.1.1.d.4.b-3 and 3.1.1.d.4.c-3 among others.

(b) Regeneration of Record Sequence Numbers.

This requirement is not met for the sample Type I system, as mentioned previously, but could be met by the software to be described later.

Example 3.1.1.d.5.b.1: Regeneration of Record Sequence Numbers using a Type II system.

The Type II systems automatically resequence record numbers to reflect addition or deletion of records. Figure 3.1.1.d.5.b-1 shows a revision of Figure 3.1.1.d.5.a-2 with the sequence numbers adjusted to reflect the addition of a line (new line 070) of data.

Example 3.1.1.d.5.b.2: Regeneration of Record Sequence Numbers using a Type III System.

This system allows for regeneration of line sequence numbers in two ways. If a file is stored without its sequence numbers and the S option is specified in the EDIT command, then the record sequence numbers of the restored file are regenerated. The defaults for this sequencing are 100 for beginning line number and an increment value of 10, as shown in the sample below.

The second means of record number regeneration is the command RESEQ, which takes as parameters the line number of the first line in the file and desired increment value. Use of the RESEQ command is also illustrated in the following sample dialogue.

SAMPLE	DATA VP/CSS	P ---	ID=GAERTNER NATIONAL CSS, INC. (STAMFORD DATA CENTER)	12/14/76	09.51.17	PAGE 1		
67.772	6.114	26.738	5.401	91.762	1.962	85.912	97.815	13.682RS100010
1.754	87.389	8.546	64.769	11.706	87.312	16.518	25.301	85.146RS100020
83.169	32.702	47.698	91.808	21.662	3.697	27.227	30.085	35.467RS100030
42.035	33.011	19.752	21.411	50.698	11.491	12.662	72.554	21.364RS100040
75.197	58.908	76.672	29.861	89.117	65.051	03.652	68.350	67.228RS100050
68.224	24.289	51.714	91.687	84.696	82.991	35.682	67.175	81.911RS100060
80.893	84.156	27.901	79.996	73.872	23.268	74.758	39.132	61.973RS100070
19.649	60.140	83.998	62.734	20.419	57.910	63.687	60.930	92.400RS100080
0.030	4.580	73.210	98.039	29.341	93.703	98.147	45.554	90.005RS100090
30.040	70.194	50.807	73.098	81.321	30.045	48.379	19.866	83.783RS100100
23.903	89.376	21.128	22.384	44.152	63.452	83.346	29.009	23.938RS100110
82.547	79.842	36.122	98.158	63.849	99.674	23.402	43.347	49.467RS100120
6.674	94.841	6.984	0.331	21.132	23.814	52.689	1.809	36.448RS100130
3.613	91.942	18.534	94.627	40.956	84.094	35.964	58.939	29.958RS100140
49.296	26.151	13.240	44.081	45.327	75.229	43.433	83.538	10.328RS100150
10.126	67.803	15.679	83.850	61.991	17.293	45.839	19.398	3.838RS100160
48.450	56.156	0.986	99.416	91.517	49.860	75.505	4.288	46.187RS100170
36.527	15.480	46.138	37.508	9.904	21.254	39.289	44.450	13.102RS100180
78.560	53.445	13.627	0.760	81.914	84.650	70.670	62.173	37.003RS100190
62.406	41.764	89.394	54.486	31.368	97.835	4.702	47.696	43.860RS100200

Figure 3.1.1.d.5.a-1. Sample data file showing line sequence numbers produced on a Type II system.

SAMPLE POKTIAN P IU=GAERTNER 11/24/76 14.13.18 PAGE 1
 VP/CSS --- NATIONAL CSS, INC. (STAMFORD DATA CENTER)

SUBROUTINE MISPTS

W.W. GAERTNER RESEARCH INC.
 1492 HIGH RIDGE ROAD
 STAMFORD, CT. 06903
 TELE (203) 322-7601

***** COMPANY PRIVATE *****

AUTHOR: W. SCHREYER

THIS ROUTINE CALCULATES THE FAILURE RATE OF MISCELLANEOUS PARTS
 IT IS AN IMPLEMENTATION OF MIL-HDBK-217B SECTION 2.13
 ALL DATA IS RECEIVED IN COMMON FROM THE CALLER
 THE FAILURE RATE IS RETURNED VIA COMMON

DECLARE TYPES

INTEGER GUBIN, SPACFL, GUFIX, GUMOB
 1 ,NAVSHL, NAVUSH, AIRINH, AIRUNH
 2 ,MISLUN
 3 ,PANFA, PAKINC, SCREEN

C

SAM000010
 SAM000020
 SAM000030
 SAM000040
 SAM000050
 SAM000060
 SAM000070
 SAM000080
 SAM000090
 SAM000100
 SAM000110
 SAM000120
 SAM000130
 SAM000140
 SAM000150
 SAM000160
 SAM000170
 SAM000180
 SAM000190
 SAM000200
 SAM000210
 SAM000220
 SAM000230
 SAM000240
 SAM000250
 SAM000260

Figure 3.1.1.d.5.a-2. Sample source code file containing automatically generated line sequence numbers produced on a Type II system.

SAMPLE2	DATA VP/CSS	P	ID=GAERTNER NATIONAL CSS, INC. (STAMFORD DATA CENTER)	12/14/76	09.31.26	PAGE 1		
07.772	6.114	26.738	5.401	91.762	1.962	85.912	97.815	13.6R2PS100010
1.754	87.389	8.546	64.769	11.706	87.312	18.518	25.301	85.146RS100020
85.169	32.702	47.688	91.808	21.662	3.607	27.227	30.085	35.467PS100030
42.035	33.011	19.752	21.411	50.698	11.491	12.662	72.554	71.364RS100040
75.197	58.908	76.672	29.861	89.117	65.951	93.652	68.350	67.228RS100050
88.224	24.289	51.714	91.687	84.696	82.991	35.682	67.175	81.911PS100060
86.893	84.156	22.901	79.996	73.872	23.268	74.758	10.132	61.972RS100070
19.644	60.140	83.998	62.734	20.419	57.910	63.687	60.930	92.400RS100080
6.030	4.590	73.210	98.023	29.341	95.703	98.147	45.554	90.005RS100090
30.040	70.144	50.807	73.098	81.321	30.045	48.370	19.866	83.783RS100100
23.903	89.376	21.128	22.384	44.152	63.452	83.346	29.009	23.938PS100110
82.547	79.842	36.122	98.158	63.844	99.674	23.402	43.347	49.467RS100120
6.674	94.841	9.984	0.331	21.133	23.814	52.680	1.800	36.648RS100130
3.613	91.842	18.534	94.627	40.956	84.094	35.964	59.939	29.959PS100140
49.240	26.151	13.240	44.081	45.327	75.229	43.433	83.538	10.328RS100150
10.126	67.823	15.679	83.850	61.991	17.293	45.839	19.398	3.838RS100160
48.450	56.156	0.886	99.916	91.517	49.860	75.505	4.288	46.187RS100170
38.527	15.480	46.138	37.508	9.804	21.254	39.289	44.450	13.102RS100180
78.560	53.445	13.627	0.760	81.914	84.650	70.670	62.173	37.003RS100190
62.466	41.764	88.394	54.486	31.368	97.835	4.702	47.696	43.860RS100200
33.896	9.637	46.757	2.810	96.050	51.008	41.596	90.503	68.649RS100210
97.304	06.371	21.909	34.105	7.457	37.798	54.675	17.864	70.110RS100220
54.986	28.339	31.031	31.145	7.592	65.247	23.155	51.707	1.845RS100230
45.713	57.667	34.589	88.529	19.871	22.476	56.018	33.821	98.768PS100240
88.221	40.412	48.478	27.164	26.681	15.604	53.498	80.552	1.834RS100250

Figure 3.1.1.d.5.b-1. Addition of data to file shown in Figure 3.1.1.d.5.a-1 to show regeneration of line sequence numbers on a Type II system.

COMMAND- ATTACH, S, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 004

COMMAND- EDITOR

..E, S, S

..L, A

100= 0.052 0.412 1.730 6.674 24.472

110= 86.765 0.341 21.157 23.878 52.854

120= 2.221 37.637 5.838 96.291 25.208

..RESEQ, 30, 30

..L, A

30= 0.052 0.412 1.730 6.674 24.472

60= 86.765 0.341 21.157 23.878 52.854

90= 2.221 37.637 5.838 96.291 25.208

..S, X, N

..BYE

COMMAND- CATALOG, X, SAMPLEDATA, ID=GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN= SAMPLE DATA

CT CY= 005 0000128 WORDS.:

COMMAND-

Figure 3.1.1.d.5.b-2. Regeneration of line sequence numbers using a Type III system.

(c) Addition of Records without Resequencing

Since resequencing is not a normal feature of some Type I systems, those systems would automatically fulfill this requirement.

Example 3.1.1.d.5.c.1: Addition of Records without Resequencing using a Type II System.

In the commercial time-sharing systems examined, the sequencing/resequencing feature can be turned off by a simple command (e.g., SER(NO) given upon entering the editor). When automatic record sequence numbering is prohibited, lines added to an existing file are simply not numbered, as in Figure 3.1.1.d.5.c-1. The editing of a file while under this prohibition appears in Figure 3.1.1.d.5.c-2.

Example 3.1.1.d.5.c.2: Addition of Records without resequencing using a Type III system.

Records may be added to files in this system without resequencing all numbers; however, this may be done only if the number of records inserted is less than the integer difference of the two lines between which the records are to be inserted. The new lines do receive sequence numbers, as the system has no other way to handle record addressing within a file. If the file is stored without the sequence numbers, then all line numbers will be automatically resequenced the next time the file is edited. An example of addition without resequencing may be found in Figure 3.1.1.d.4.a-3.

Program Requirements for Necessary New Support Software:

Input: Any fixed-record-length file.

Parameters (Optional):

- (1) Number of first record in file
- (2) Incremental value
- (3) 3-letter identifier for file.

If any parameters are lacking, system defaults go into effect.

SAMPLE3	DATA VP/CSS	P ---	ID=GAERTNER NATIONAL CSS, INC.	12/14/76 (STAMFORD DATA CENTER)	09.51.49 PAGE 1				
67.772	6.114	26.738	5.401	91.762	1.962	85.912	97.815	13.682	RS100010
55.555	3.023								
1.754	87.389	8.546	64.769	11.706	87.312	16.518	25.301	85.146	RS100020
83.169	32.702	47.688	91.808	21.662	3.697	27.227	30.085	35.467	RS100030
42.035	33.011	19.752	21.411	50.698	11.491	12.662	72.554	71.364	RS100040
75.197	58.908	76.672	29.861	89.117	65.051	93.652	68.350	67.228	RS100050
88.224	24.289	51.714	91.687	84.696	82.991	35.682	67.175	81.911	RS100060
80.893	84.156	22.901	79.996	73.872	23.268	74.758	39.132	61.973	BS100070
19.649	60.140	83.998	62.734	20.419	57.910	63.687	60.930	92.400	RS100080
6.030	4.580	73.210	98.038	29.341	93.703	98.147	45.554	90.005	BS100090
30.040	70.194	50.807	73.098	81.321	30.045	48.379	19.866	83.783	RS100100
23.903	80.376	21.128	22.384	44.152	63.452	83.346	29.009	23.938	RS100110
62.547	79.842	36.122	98.158	63.849	99.674	23.402	43.347	49.467	RS100120
6.674	94.841	8.984	0.331	21.133	23.814	52.689	1.809	36.648	RS100130
3.613	91.842	18.534	84.627	40.956	84.094	35.964	58.939	29.958	RS100140
49.296	26.151	13.240	44.081	45.327	75.229	43.433	83.538	10.328	BS100150
10.126	67.803	15.670	83.850	61.991	17.293	45.839	19.398	3.838	RS100160
48.450	56.156	0.886	99.916	91.517	49.860	75.505	4.288	46.187	BS100170
38.527	15.480	46.139	37.508	9.804	21.254	39.289	44.450	13.102	BS100180
78.560	53.445	13.627	0.760	81.914	84.650	70.670	62.173	37.003	RS100190
62.466	41.764	88.394	54.486	31.368	97.835	4.702	47.696	43.860	BS100200

Figure 3.1.1.d.5.c-1. Listing of file from a Type II system that shows addition of data without regeneration of line sequence numbers.

```

13.26.42 >EDIT SAMPLE DATA
EDIT:
>SER (NO)
>N 2      1.754 87.389 8.546 64.769 11.706 87.312 18.518 25.301 85.146
BS100020
>I 55.555 3.023
>TOP 3
67.772 6.114 26.738 5.401 91.762 1.962 85.912 97.815 13.682
BS100016
55.555 3.023
1.754 87.389 8.546 64.769 11.706 87.312 18.518 25.301 85.146
BS100020
>FILE SAMPLE3 DATA
KLING: SAMPLE3 DATA P
13.28.57 >

```

Figure 3.1.1.d.5.c-2. Demonstration of command on a Type II system to add a line of data to an existing unit without regeneration of line sequence numbers.

Program Performance:

- (1) Determine which parameters were included in program call.
- (2) Put into effect necessary defaults and initialize values.
- (3) Step through each record of file, checking last 8 characters.
 - (a) If last 8 characters begin with 3 letter identifier, then resequence accordingly.
 - (b) If last 8 characters are blank, then sequence accordingly.
 - (c) If last 8 characters contain other data, issue error message and terminate.

Output: An edited file with 3 letter identifier and 5 digit sequence number in last 8 columns.

Requirement 3.1.1.d.6

Classification: Basic

Requirement:

Copy one or more records from one library data file to another library data file:

- (a) Within a single library.
- (b) Between libraries.

Purpose of Paragraph:

To define inter-file record copying requirements.

Examples:

Each of the following three examples solve the same problem: That of copying a record (line 3 is chosen) from one file into another file. The copied record will become the fourth line of the new file. It is assumed that the files involved are resident on random access storage devices, though they may have been copied there from another medium.

Example 3.1.1.d.6.1: Copying one or more records between files using a Type I system.

In order to copy one or more records from file A to file B on this system, file A is accessed as read only and the desired record(s) copied into a temporary buffer. File B is then accessed with full editing capabilities, the pointer moved to the desired point of record insertion, and the records retrieved from the temporary buffer. If more than one file is to be either source or destination of the copied records, this procedure is repeated. Whether the files exist in the same or different libraries has no effect, as files are assumed to have been moved to random access storage and are available to the editor by issuing their file designations.

In the sample dialogue of Figure 3.1.1.d.6-1, file A, SAMPLE.DAT, is accessed via the ER (EDIT READ) command, which does not make a working copy of the file. As before, the R\$3AV\$\$ commands are issued to read the file into the editor's buffer, advance the pointer and display the current line. That line is then saved into a temporary buffer via the SAVE (S) command. If more than one record were to be saved, then the command would require as parameter the total number of contiguous records to be saved.

The edit buffer is then cleared by moving the pointer to the beginning of the file and erasing it entirely (B/K\$). At this point the destination file is accessed and prepared for full editing. The place of record insertion is indicated by moving the pointer to line 4, and the record retrieved from the temporary buffer via the UNSAVE command. This command takes no parameters, as it always copies the complete contents of the temporary buffer. As usual, file alteration may be verified by displaying the entire file.

```

•R E
•ERSAMPLE.DATSR$3AV$$
 18.246 41.567 85.188 37.022 55.437
•SS$
•B/K$$
•ERSAMPLE.DATSR$$
•4ASVSU$$
 84.626 80.885 23.679 14.110 71.548
•B/L$$

 0.082 0.412 1.730 6.674 24.472
 66.765 0.341 21.157 23.878 52.854
 2.221 37.637 5.838 96.291 25.208
 18.246 41.567 85.188 37.022 55.437
 84.626 80.885 23.679 14.110 71.548
•EX $$

```

Figure 3.1.1.d.6-1. Copying a record from one library data file to another library data file on a Type I system.

Example 3.1.1.d.6.2: Copying one or more records between files using a Type II system.

Commercial time-share systems are typically more flexible and powerful than the other systems, which is once again shown by the inter-file copying capability. Here, the entire source file does not have to be retrieved. Instead, the destination file is entered for editing, the pointer placed at the desired insertion point, and a GET command issued.

Parameters to this command indicate (1) the source file, (2) beginning record number and (3) ending record number of the records to be retrieved. If only one record is to be fetched, then its sequence number is given as both parameters two and three, as is shown in Figure 3.1.1.d.6-2. As in the Type I system, library designation is contained in the file type, and has no effect on the GET command.

```
13.45.47 >EDIT SAMPLE DATA
EDIT:
>N 3
    2.221  37.637   5.838  96.291  25.208
>GET SAMPLE DATA1 3 3
    18.246  41.567  85.188  37.022  55.437
>T/P 9
    0.082   0.412   1.730   6.674  24.472
    86.765  0.341  21.157  23.878  52.854
    18.246  41.567  85.188  37.022  55.437
    2.221  37.637   5.838  96.291  25.208
    84.626  80.885  23.679  14.110  71.548
>FILE
13.47.32 >
```

Figure 3.1.1.d.6-2. Copying a record from one library data file to another library data file utilizing a Type II system editor.

Example 3.1.1.d.6.3: Copying one or more records between files using a Type III system.

When working with such a limited system as this, the user must construct a working copy from both source and destination files by editing both alternately. This working copy is then re-edited to eliminate unwanted characters caused by the file merging, then stored in the usual manner.

The sample dialogue below shows that both source and destination files are accessed simultaneously and assigned the local file names of S1 and S2, respectively. Since the desired output file would consist of the first three lines of S2, the third line of S1, followed by the remainder of S2, it is necessary to first copy records 1-3 of S2. This is done by saving lines numbered 100 through 120 into a work file called X, using the no sequence and merge options (S,X,N,M,100,120). Next, the source file is edited and its third record saved into file X in a similar manner. Finally, the remainder of S2 is copied into the work file. S2 must be rewound prior to editing it a second time.

COMMAND- ATTACH, S1, SAMPLEDATA1, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- ATTACH, S2, SAMPLEDATA2, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- EDITOR

..E, S2, S

..L, 100, 120

100= 0.427 1.685 6.262 22.412 78.113

110= 66.969 98.796 90.053 51.151 96.434

120= 18.246 41.567 85.188 37.022 55.437

..S, X, N, M, 100, 120

..E, S1, S

..L, 120

120= 2.221 37.637 5.838 96.291 25.208

..S, X, N, M, 120

..REWIND, S2

..E, S2, S

..L, 130, L

130= 99.429 97.641 90.982 67.123 83.901

..S, X, N, M, 130, L

..E, X, S

..L, /*EOR/, A

130=*EOR

150=*EOR

..D, 130

..D, 150

..L, A

100= 0.427 1.685 6.262 22.412 78.113

110= 66.969 98.796 90.053 51.151 96.434

120= 18.246 41.567 85.188 37.022 55.437

140= 2.221 37.637 5.838 96.291 25.208

160= 99.429 97.641 90.982 67.123 83.901

Figure 3.1.1.d.6-3. Copying a record from one library data file to another library data file on a Type III system.

..S,Y,N

..BYE

COMMAND- CATALOG,Y,SAMPLEDATA2,1D-GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT 1D= GAERTNER PFN=SAMPLEDATA2

CT CY= 002 00000128 WORDS.:

COMMAND-

Figure 3.1.1.d.6-3 (continued). Copying a record from one library data file to another library data file on a Type III system.

3.1.2 Full Source Data Maintenance Requirements

Classification: Full

Requirement:

Full Source Data Maintenance requirements include all of the Basic Source Data Maintenance requirements and the following additional requirements.

Purpose of Paragraph:

To define which paragraphs constitute Full implementation of Source Data Maintenance requirements.

3.1.2.a Data File Storage

Classification: Basic

Requirement:

Data File Storage - Provision must be made for storing data (except for object and load modules) in a compressed form to minimize storage space. This requirement involves:

1. The compression of data being stored to eliminate leading and trailing blank characters in each record, as well as intervening blanks when more than three appear contiguously. This will result in storage of significant characters only.
2. Restoration of data from its compressed library storage form to its original uncompressed form.

Purpose of Paragraph:

To state and define compressed storage requirements for data files other than object and load modules.

Existing Tools which Satisfy this Requirement:

Editors which include storage options concerning data compression would fulfill this requirement. Not all systems have editors which contain such options.

Examples:

- 1) Compression of data for storage.

Since source files formatted in accordance with TDSP usually contain leading, trailing and intervening strings of blanks, storage is inefficient. Since facilities for compressing files are either widespread or easily implemented, it is suggested that this requirement be listed as Basic rather than Full.

Example 3.1.2.a.1.1: Compression of data for storage using a Type I system.

In this system, leading blanks may be suppressed for storage by specifying certain tab settings during editing. As the system now stands, trailing blanks are also eliminated. However, if record sequence numbering is to be employed as required, then this elimination would no longer take place since the last eight characters would be used for sequencing. There therefore would be no trailing blanks. Instead, a string of intervening blanks would appear in the record. It is suggested that the post-processor proposed for serialization also include a PACK option which would remove all unnecessary blanks, probably utilizing the same techniques employed by the sample Type II system.

Figure 3.1.2.a.1-1 shows how the tab settings are employed to eliminate leading blanks.

The "!" is used to represent the tab character. Since the default settings of the tab are every eight characters, and this Editor has no way of altering the default, the net effect will be to move the source code one space to the right. However, the storage cost of these leading blanks is only the one tab character, thus saving three bytes of storage in just three lines. Note the use of the macro facility (discussed more fully in Example 3.1.2.b.1.b.1) to implement the file-wide alteration. In it, the pointer is advanced 6 spaces to just before the text, then all characters (which happen to be blanks) from the beginning of the line up to the pointer are replaced with the tabs character (OC!)

```
.R EDIT
*EBSAMPLE.FOR$I$/L$$
  BASEA=CDEC+BSUMA
  BASEAA=(WSTRL+E!TRL+YQAST)/PI
  BESTFT=BESTFT+BASEA+2.*BASEAA
  GAUSIV=FGAUS(BESTFT)
*1/6J$OC!$A$/$$
*B$4E1$$
*B$/L$$
  BASEA=CDEC+BSUMA
  BASEAA=(WSTRL+E!TRL+YQAST)/PI
  BESTFT=BESTFT+BASEA+2.*BASEAA
  GAUSIV=FGAUS(BESTFT)
*E$$$
```

Figure 3.1.2.a.1-1. Compression of data for storage in a Type I system.

Example 3.1.2.a.2: Compression of data for storage using a Type II system.

The commercial time-share systems, naturally being economical of space, all include data compression facilities. In the CSS system, compression is an editor option applied to the output file via OUTBY PACK (see Figure 3.1.2.a-2).

This requires actually entering the editor, though only to issue the FILE command. The system automatically appends a "P" to the file type designation to distinguish storage format. Note that in the example given, a 13 to 1 reduction in space has been achieved.

The CMS system does not require the user to enter the editor to compress TEXT. Instead, the COPYFILE command is used with the PACK option. This system offers the further advantage of allowing the user to specify the most frequent character if it is a non-blank, thus increasing the flexibility of the compression.

In either system, the more blank strings (or strings of specified character) occurring in the file, the greater the storage saving. It should be noted that if there are very few occurrences of replaceable strings, the PACK option might increase the file size.

```
13.32.47 >EDIT SAMPLE DATA OUTBY PACK
EDIT:
>FILE
```

```
13.33.04 >LIST SAMPLE DATA*
FILENAME FILETYPE MODE ITEMS
SAMPLE DATA P 13
SAMPLE DATAP P 1
```

```
13.33.42 >
```

Figure 3.1.2.a-2. Demonstration of file compression on a Type II system.

Example 3.1.2.a.3: Compression of data for storage using a Type III system.

The sample Type III system, the CDC 6600 with Intercom, does not currently include either compression or restoration facilities. The only means of reducing storage space is to file data without sequence numbers. While this eliminates trailing blanks, it is not sufficient to meet the compression requirement. Additional software would be needed for such systems.

2) Restoration of data from compressed form.

Example 3.1.2.a.2.1: Restoration of data from compressed storage using a Type I system.

As was noted in Example 3.1.2.a.1.1, actual data compression does not occur, therefore restoration is unnecessary. In the proposed system, however, a post-processor would sequence and, if desired, pack the records. Therefore, modifications would have to be made to the editor or a pre-processor for packed files supplied in order to unpack compressed files. The latter option is the simplest, though some minor modification of the editor would be required to warn the user if he attempts to edit a packed file.

Example 3.1.2.a.2.2: Restoration of data from compressed storage using a Type II system.

In CSS, restoration of data is done simply by specifying the INBY UNPACK option of the EDIT command. As shown in Figure 3.1.2.a.2-2, the user may specify that the file be unpacked for editing only (first part of example), or may be saved in unpacked form as shown in the second part of the example.

The CMS system requires a specific command to restore a file: COPYFILE UNPACK.

EXAMPL44 MEMO P ----- 20APR79 ----- PAGE 1

```
22.39.26 >listf sample data*
SAMPLE  DATAP  P          1
```

```
22.39.38 >edit sample data inby unpack
EDIT:
>file
```

```
22.40.28 >listf sample data*
SAMPLE  DATA  P          13
SAMPLE  DATAP  P          1
```

```
22.40.34 >erase sample datap
```

Figure 3.1.2.a.2-2. Restoration of data from compressed form using a Type II system.

Program Requirements for Necessary New Support Software

1) Type I System:

(a) Post-processor

Modify system proposed in paragraph 3.1.1.d.5 such that input is any fixed-record-length file in unpacked format. Include a fourth optional parameter, PACK, to request compression. Include a fourth item under program performance: Eliminate leading and internal strings of blanks, replacing them with a two byte code. Trailing blanks would be eliminated and an end-of-record mark inserted if needed. It could be necessary to modify file format to include a record or header which identifies the file as packed, and specifies the unpacked record length. This header would be maintained by the post-processor.

(b) Pre-Processor

Input: A source file post-processed with the PACK option.

(1) Check that file is packed and issue error message if it is not.

(2) Decode each record such that leading and intervening blanks are restored.

(3) Put blanks at end to bring record length to that designated for file.

(4) Alter the header record to identify the file as unpacked.

Output: An image of the file as it was prior to packing during post-processing.

(c) Editor Alterations

The editor would need to be modified as follows:

(1) File format now includes additional header information. The editor must be able to accept this.

(2) Editor should check that the file is unpacked, and issue an error message if it is not.

2) Type III System

(a) Pre-processor specifications would be the same in a Type III system as in the Type I system.

(b) Post-processor would be designed solely to pack a file.

Input: Unpacked source file.

Program Performance:

(1) Check that file is unpacked and issue error message if it isn't.

(2) Eliminate leading and intervening strings of blanks, replacing them with a two byte code. Eliminate trailing blanks and insert end-of-record mark.

(3) Alter header to identify file as packed and to indicate record length.

Output: Packed version of input file with some name change made to indicate file status.

(c) Editor would require modification so that it accepts files with new format and checks file status.

3.1.2.b Full Data Maintenance

Classification: Full

Requirement:

Full Data Maintenance - A PSL must provide for the generation and updating of library files. In addition to the basic requirements there must also be the following capabilities:

(1) Make the following changes to a file of source data as well as to a data file:

(a) Modify a record of data (i.e., change one or more characters in a record without requiring the user to input the entire line).

- (b) Scan for and replace a specific string of data in a single record or in every record in a data file in which that string appears.
 - (c) Insert a string of data into specific positions of every record of a data file.
- (2) Make temporary changes to a data file (i.e., change a data file for the duration of a single job without making permanent changes to the stored data file).
- (3) Automatically generate program stubs for subroutines and coroutines which are referenced in the source code currently in development but which themselves have not yet been coded. A program stub is defined as an incomplete routine which minimally includes entry and exit points and statements causing printout of the name of the routine entered.
- (4) Concatenate two or more PSL data files from two or more different PSL libraries into a single file without destroying the integrity of the source files.

Intent of Paragraph:

To define those capabilities beyond the basic level which constitute a Full implementation of PSL data maintenance procedures.

Existing Tools which Satisfy This Requirement:

Paragraph 3.1.2.b.1: The more sophisticated editors currently available offer these capabilities.

Paragraph 3.1.2.b.2: By using standard system utilities for duplication of files and standard editing facilities for creating the desired changes, this requirement may be fulfilled by almost all existing systems.

Paragraph 3.1.2.b.3: No known widely used commercial systems currently include the capability to automatically generate program stubs.

Paragraph 3.1.2.b.4: Merge routines are frequently supplied as system utilities along with other system software. Such standard merge facilities fully satisfy this requirement.

Examples:

Example 3.1.2.b.1.a: Modification of a line of source data.

Since all known Editors may handle either data or source files, the same facilities which satisfied requirement 3.1.1.d.3 also satisfy this requirement. The following figures show how characters of source data may be replaced using the same commands as were used in editing data files in Examples 3.1.1.d.3.1, 3.1.1.d.3.2 and 3.1.1.d.3.3. If modification is interpreted to include addition and deletion of characters within a record of a source file, then the commands illustrated in Examples 3.1.1.d.1.1, 3.1.1.d.1.2, 3.1.1.d.1.3, 3.1.1.d.2.1, 3.1.1.d.2.2 and 3.1.1.d.2.3 fill this requirement as well. The reader should reference those examples when reviewing the sample dialogues which follow.

The only item within those dialogues which differs from the preceding ones is in the Type II system example. There the reader may notice that additional parameters have been included in the change command following the delimited strings. The first of these parameters designates the number of lines (including the current line) to which the command is to apply. The second parameter is the number of occurrences within each line to which the command applies. Thus the command "C /H/ROL/ 2 1" indicates that in the current line and the one following it, the first occurrence of an H should be replaced with the string ROL.

Example 3.1.2.b.1.b.1: Scanning for and replacement of a source data string in one or all records in a file using a Type I system.

The sample dialogues of Figures 3.1.1.d.1-1 and 3.1.1.d.2-1 illustrate string replacement in one record of a data file. The same commands are used to achieve the same outcome for a source file. Replacement of all occurrences of a specific string in a file, either source or data, may be accomplished by building the desired replacement into a macro, then executing the macro on as many records of the file as is necessary. This method offers the further flexibility of string replacement on more than one but not all file records.

In Figure 3.1.2.b.1.b-1 the editor is entered as in preceding examples, the file read into the buffer and listed. Then a macro is created to replace IPOINT with JPTPL1.

M/GIPOINT \$=CJPTPL1\$/\$\$

The pointer is moved to the beginning of the file and macro executed on the following 99 lines. When the system cannot locate all 99 lines (the file contains only 3 lines) it issues the message "?*SRCH FAIL IN MACRO*?" The alterations are then verified as usual.

```
.R E
*EBSAMPL.FORSRS/LSS
    A=B(IPOINT) +C(JPOINT)
    A2=B(IPOINT)*B(IPOINT)
    A3=B(IPOINT)*B(IPOINT)*B(IPOINT)
*M/GIPOINTS=CJPTPL1$/$$
*BS99EMSS
?*SRCH FAIL IN MACRO*?
*B/LSS
    A=B(JPTPL1) +C(JPOINT)
    A2=B(JPTPL1)*B(JPTPL1)
    A3=B(JPTPL1)*B(JPTPL1)*B(JPTPL1)
*EX SS
```

Figure 3.1.2.b.1.b-1. Scanning for and replacing a specific string of data in every line in a file using a Type I system editor.

Example 3.1.2.b.1.b.2: Scanning for and replacement of a source data string in one or all records in a file using a Type II system.

Typical time-sharing systems offer the same replacement scope and flexibility while requiring only a single command: The CHANGE command with additional parameters. As mentioned previously, the first parameter indicates the number of (contiguous) lines for search and replacement, while the second indicates the number of occurrences within each line. An asterisk in either position means all lines or occurrences. The sample dialogue (Figure 3.1.2.b.1.b-2) causes the whole file to be printed, then makes the desired replacement. Since verification is automatic, the altered file may then be stored. The entire operation took little more than 1½ minutes.

```

14.23.13 >EDIT SAMPLE2 FORTRAN
EDIT:
>P 5
      A=B(IPOINT) +C(JPOINT)
      A2=B(IPOINT)*B(IPOINT)
      A3=B(IPOINT)*B(IPOINT)*B(IPOINT)
* EOF
>T
>C /IPOINT/JPTPL1/ * *
      A=B(JPTPL1) +C(JPOINT)
      A2=B(JPTPL1)*B(JPTPL1)
      A3=B(JPTPL1)*B(JPTPL1)*B(JPTPL1)
* EOF
>FILE

# 24.45 >

```

Figure 3.1.2.b.1.b-2. Scanning for and replacing a specific string of data in every line of a file using a Type II system editor.

Example 3.1.2.b.1.b.3: Scanning for and replacement of a source data string in one or all records in a file using a Type III system.

Like the Type II system, the standard CHANGE command is modified to alter all occurrences of a string in a file. This system, however, does not offer the additional flexibility of altering less than all file records, nor does it ever allow less than all occurrences of the specified change per record. As shown below, the basic replacement command operates on all records on a file when it includes the A specification following the replacement string.

COMMAND- ATTACH, S, SAMPLE2FORT, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- EDITOR

..E,S,S

..L,A

100= A=B(IPOINT) +C(JPOINT)

110= A2=B(IPOINT)*B(IPOINT)

120= A3=B(IPOINT)*B(IPOINT)*B(IPOINT)

.. /IPOINT/= /JPTPL1/, A

6 CHANGES

..L,A

100= A=B(JPTPL1) +C(JPOINT)

110= A2=B(JPTPL1)*B(JPTPL1)

120= A3=B(JPTPL1)*B(JPTPL1)*B(JPTPL1)

..S,X,N

..BYE

COMMAND- CATALOG,X, SAMPLE2FORT, ID=GAERTNER

NEW CYCLE CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN=SAMPLE2FORT

CT CY= 002 0000128 WORDS.:

COMMAND-

Figure 3.1.2.b.1.b-3. Scanning for and replacing a specific string of data in every line in a file using a Type III system.

Example 3.1.2.b.1.c.1: Insertion of a string of data into specific positions in every record using a Type I system.

Once again, the macro creation facility fulfills the requirement. The sample below shows the typical initial steps, after which the macro is set up. In it, the written-line character pointer is advanced 5 characters to position 6 (5J\$) then the next 3 characters are specified as the field of operation (3D\$). Finally, the insert command is followed by the desired string, the line verified, and the pointer advanced to the next record. The macro is actually executed on four lines.

```

.R E
*EBSAMPLE.DAT$R$/L $$

0.082  0.412  1.730  6.674  24.472
86.765  0.341  21.157  23.878  52.854
2.221  37.637  5.838  96.291  25.208
84.005  80.885  23.679  14.110  71.548
*M/5J$3D$1005$V$AS/$$
*4EM$$

0.005  0.412  1.730  6.674  24.472
86.005  0.341  21.157  23.878  52.854
2.005  37.637  5.838  96.291  25.208
84.005  80.885  23.679  14.110  71.548
*EX $$

```

Figure 3.1.2.b.1.c-1. Inserting a string of data into specific positions of every line of a file by way of a Type I system editor.

Example 3.1.2.b.1.c.2: Insertion of a string of data with specific positions in every record using a Type II system.

Most commercial time-sharing systems offer a command specifically designed to perform this function. The OVERLAY command specifies the string to be inserted; the placement of insertion is specified by the number of blanks preceding the specified string. These blanks are not overlaid on the existing text. The example of Figure 3.1.2.b.1.c-2 shows the overlay defined and applied to the current line, then applied to the succeeding three lines by use of the AGAIN command.

The sample Type III system examined did not include any commands which could fulfill this requirement, nor could a simple additional routine solve the problem easily. Unfortunately, the only satisfactory solution would be modification of the editor itself.

```

B.51.41 >EDIT SAMPLE DATA
EDIT:
>P 9
    0.082    0.412    1.730    6.674    24.472
    86.765    0.341    21.157    23.878    52.854
    2.221    37.637    5.838    96.291    25.208
    84.626    80.885    23.679    14.110    71.548
*EOF
>T
>O      005
    0.005    0.412    1.730    6.674    24.472
>A 3
    86.005    0.341    21.157    23.878    52.854
    2.005    37.637    5.838    96.291    25.208
    84.005    80.885    23.679    14.110    71.548
>FILE
15.53.49 >

```

Figure 3.1.2.b.1.c-2. Inserting a string of data into specific positions of every line of a file utilizing a Type II system.

Paragraph 3.1.2.b.2:

In the following examples, temporary changes are accomplished by renaming the original file to a save name, renaming the file containing the temporary changes so that they now have the name of the original file, then executing the program as usual. The renaming process is reversed afterward to restore the original file designation. Use of this technique makes the following assumptions:

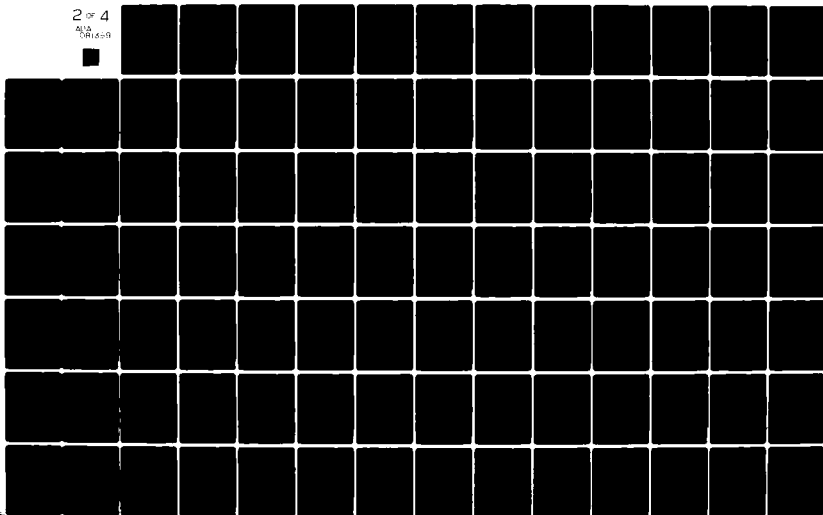
- (a) The desired temporary change is in the data. However, the same technique could also be used if the program itself were the variable.
- (b) The program which uses the data calls it by a specific name, that of the original name of the original file.
- (c) The desired temporary replacement data is currently stored with a known name.

AD-A081 389

GAERTNER (W W) RESEARCH INC STAMFORD CONN
PROGRAMMING SUPPORT LIBRARY, VOLUME II, GUIDELINES FOR IMPLEMENT--ETC(U)
NOV 79 C M TURCIO, W W SCHREYER, N A ADAMS F30602-78-C-0103
RADC-TR-79-241-VOL-2 NL

UNCLASSIFIED

2 of 4
ALIA
ORIGIN



Example 3.1.2.b.2.1: Making temporary changes to a file using a Type I system.

The above-mentioned methods are implemented by using the Peripheral Interchange System. In Figure 3.1.2.b.2-1 the program PROGRAM is run to show the original data. Then, by running PIP, the respective files are renamed: FTN1.DAT is saved as SAVFT1.DAT, and the temporary data file TEMP.DAT renamed FTN1.DAT. PIP is then exited, and PROGRAM rerun with the new (temporary data). Finally PIP is entered again and the renaming process reversed.

```

.R PROGRAM
  INPUT DATA

    0.082    0.412    1.730    6.674    24.472
    86.765   0.341   21.157   23.878   52.854
    2.221   37.637    5.838   96.291   25.208
    84.626   80.885   23.679   14.110   71.548

STOP--

.R PIP
*SAVFT1.DAT/R=FTN1.DAT
*FTN1.DAT/R=TEMP.DAT
*!C

.R PROGRAM
  INPUT DATA

    0.427    1.685    6.262   22.412   78.113
    66.969   98.796   90.053   51.151   96.434
    18.246   41.567   85.188   37.022   55.437
    99.429   97.641   90.982   67.123   83.901

STOP--

.R PIP
*TEMP.DAT/R=FTN1.DAT
*FTN1.DAT/R=SAVFT1.DAT
*!C

```

Figure 3.1.2.b.2-1. Making temporary changes to a file in a Type I system.

Example 3.1.2.b.2.2: Making temporary changes to a file using a Type II system.

In most commercial time-sharing systems file renaming is done by a simple system command; CSS uses the ALTER command while CMS uses the RENAME command. Both operate similarly. As before, Figure 3.1.2.b.2-2 below first shows program execution and original data. That data is then renamed from FILE FT01F001 P to SAVE1 DATA P. (Note: This and other systems use three identifiers for a file: Filename; filetype; filemode, i.e., disk on which file is resident). The temporary data designation is similarly altered, the program rerun, then the file designations changed back again.

This method is slightly more convenient in that fewer commands are required than in the Type I system.

```
16.22.36 >PROGRAM
      INPUT DATA FROM UNIT 1

      0.082    0.412    1.730    6.674    24.472
      86.765    0.341    21.157    23.878    52.854
      2.221    37.637    5.838    96.291    25.208
      84.626    80.885    23.679    14.110    71.548
STOP

16.25.37 >ALTER FILE FT01F001 P SAVE1 DATA P

16.25.56 >ALTER TEMP DATA P FILE FT01F001 P

16.26.15 >PROGRAM
      INPUT DATA FROM UNIT 1

      0.427    1.685    6.262    22.412    78.113
      66.969    98.796    90.053    51.151    96.434
      18.246    41.567    85.188    37.022    55.437
      99.429    97.641    90.982    67.123    83.901
STOP

16.29.21 >ALTER FILE FT01F001 P TEMP DATA P

16.29.40 >ALTER SAVE1 DATA P FILE FT01F001 P

16.29.59 >
```

Figure 3.1.2.b.2-2. Making temporary changes to a file on a Type II system.

Example 3.1.2.b.2.3: Making temporary changes to a file using a Type III system.

The Type III system, on the other hand, requires a more complex version of the same operation due to its unwieldy manner of file handling. Here, the command is RENAME, and it requires two parameters: The original file name and the new file name. After the original and temporary data files have been renamed using this command, the file to be used for program execution must be released, and so the RETURN command is issued.

One advantage of this system is that the renaming is automatically temporary unless the files are specifically returned to storage with their new names. Therefore it is not necessary to reverse the renaming process as in the other two systems.

COMMAND- PROGRAM
INPUT DAT FROM SAMPLE DATA

0.082	0.412	1.730	6.674	24.472
86.765	0.341	21.157	23.878	52.854
2.221	37.637	5.838	96.291	25.208
84.626	80.885	23.679	14.110	71.548

0.743 CP SECONDS EXECUTION TIME
COMMAND- ATTACH, S, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 001
COMMAND- ATTACH, S2, SAMPLEDATA2, ID=GAERTNER

PF CYCLE NO.= 001
COMMAND- RENAME, S, SAMPDATSAV

NM ID= GAERTNER PFN=SAMPLEDATA
NM CY= 001 0000128 WORDS.:
RN ID= GAERTNER PFN=SAMPDATSAV
RN CY= 001 0000128 WORDS.:
COMMAND- RENAME, S2, SAMPLEDATA

NM ID= GAERTNER PFN=SAMPLEDATA2
NM CY= 001 0000128 WORDS.:
RN ID= GAERTNER PFN=SAMPLEDATA
RN CY= 001 0000128 WORDS.:
COMMAND- RETURN, S2

Figure 3.1.2.b.2-3. Making temporary changes to a file on a Type III system.

COMMAND- PROGRAM
 INPUT DAT FROM SAMPLE DATA

0.427	1.685	6.262	22.412	78.113
66.969	98.796	90.053	51.151	96.434
18.246	41.567	85.188	37.022	55.437
99.429	97.541	90.982	67.123	83.901

0.747 CP SECONDS EXECUTION TIME
 COMMAND- ATTACH, S2, SAMPLEDATA, ID=GAERTNER

PF CYCLE NO.= 001
 COMMAND- RENAME, S2, SAMPLEDATA2

NM ID= GAERTNER PFN=SAMPLEDATA
 NM CY= 001 0000128 WORDS.:
 RN ID= GAERTNER PFN=SAMPLEDATA2
 RN CY= 001 0000128 WORDS.:
 COMMAND- RENAME, S, SAMPLEDATA

NM ID= GAERTNER PFN=SAMDATSAV
 NM CY= 001 0000128 WORDS.:
 RN ID= GAERTNER PFN=SAMPLEDATA
 RN CY= 001 0000128 WORDS.:
 COMMAND- RETURN, S, S2

COMMAND-

Figure 3.1.2.b.2-3 (continued). Making temporary changes to a file on a Type III system.

Paragraph 3.1.2.b.3: Automatic generation of program stubs.

As previously mentioned in the discussion of existing software tools for this paragraph, no known standard commercial system offers this facility. Therefore, additional software would be needed in order to meet the full PSL implementation requirements. The nature of this software is more fully described in the Section "Program Requirements for Necessary New Software" following the Examples section for this paragraph.

Example 3.1.2.b.4.1: Merging of two data files using a Type I system.

The basic method on this and the next two examples is to form a third file which is a merge of two other files. The original files may be resident in different libraries or the same library. They may be either data or source code. In the examples the files are resident on random access devices, but that is not necessary. Either or both could have resided on magnetic tape or any other storage media attachable to the system.

Once again, the Peripheral Interchange Program is called to perform file manipulation in this typical Type I system (Figure 3.1.2.b.4-1). After PIP is called, the directory listing and contents of file one, A.DAT, are printed, then the same information is printed for B.DAT. After the files have been thus verified, a new file, C.DAT, is created by copying the other two files into it (C.DAT=A.DAT, B.DAT). This new file is then verified in the same way as the original file.

Example 3.1.2.b.4.2: Merging two data files using a Type II system.

The commercial time share system supports the command COMBINE which performs the merge function but which requires that both files be disk resident. In the example of Figure 3.1.2.b.4-2, the user first verifies both input files, then issues the COMBINE command, which requires as parameters the designation of the created (output) file followed by the designations of all files to be included. There may be any number of input files, as long as the command itself does not exceed one input line of the user's input device. Input files must be of the same format (fixed or variable) and if fixed must have the same record length. No such limitations apply to the CMS equivalent command COPYFILE, in which the user may specify file format and record length for all files involved. The system will convert formats, truncating or padding as needed.

EXAMPL57 MEMO P ----- 20APR79 ----- PAGE 1

```
11.24.42 >listf * data
FILENAME FILETYPE MODE  ITEMS
FIRST    DATA      P      3
SECOND   DATA      P      4
```

```
11.26.09 >combine all data P first data P second data P
```

```
11.26.28 >listf * data
FILENAME FILETYPE MODE  ITEMS
FIRST    DATA      P      3
ALL      DATA      P      7
SECOND   DATA      P      4
```

Figure 3.1.2.b.4-2. Merging two files using a Type II system.

```

.R PIP
*A.DAT/L
22-NOV-76
A .DAT 1 22-NOV-76
1316 FREE BLOCKS
*TT:=A.DAT

0.427 1.685 6.262 22.412 78.113
66.969 98.796 90.053 51.151 96.434
18.246 41.567 85.188 37.022 55.437
99.429 97.641 90.982 67.123 83.901

*B.DAT/L
22-NOV-76
B .DAT 1 22-NOV-76
1316 FREE BLOCKS
*TT:=B.DAT

0.082 0.412 1.730 6.674 24.472
86.765 0.341 21.157 23.878 52.854
2.221 37.637 5.838 96.291 25.208
84.626 80.885 23.679 14.110 71.548

*C.DAT=A.DAT,B.DAT
*C.DAT/L
22-NOV-76
C .DAT 2 22-NOV-76
1314 FREE BLOCKS
*TT:=C.DAT

0.427 1.685 6.262 22.412 78.113
66.969 98.796 90.053 51.151 96.434
18.246 41.567 85.188 37.022 55.437
99.429 97.641 90.982 67.123 83.901
0.082 0.412 1.730 6.674 24.472
86.765 0.341 21.157 23.878 52.854
2.221 37.637 5.838 96.291 25.208
84.626 80.885 23.679 14.110 71.548

*IC

```

Figure 3.1.2.b.4-1. Merging two PSL data files from two different PSL libraries into a single file on a Type I system.

Example 3.1.2.b.4.3: Merging of two data files using a Type III system.

The Type III system command is also COPY. However, as shown in Figure 3.1.2.b.4-3, each input file must be individually copied into the output file and the output file then edited to delete the end of record mark(s) used to delimit files.

COMMAND- ATTACH, A, ADATA, I D=GAERTNER

PF CYCLE NO.= 001

COMMAND- ATTACH, B, BDATA, I D=GAERTNER

PF CYCLE NO.= 001

COMMAND- COPYCR, A, TC

COMMAND- COPYCR, B, TC

COMMAND- EDITOR

.. E, TC, S

.. L, A

100=	0.427	1.685	6.262	22.412	78.113
110=	66.969	98.796	90.053	51.151	96.434
120=	18.246	41.567	85.188	37.022	55.437
130=	99.429	97.641	90.982	67.123	83.901
140=	*EOR				
150=	0.082	0.412	1.730	6.674	24.472
160=	86.765	0.341	21.157	23.878	52.854
170=	2.221	37.637	5.838	96.291	25.208
180=	84.626	80.885	23.679	14.110	71.548

.. D, 140

.. S, C, N

.. BYE

COMMAND- CATALOG, C, CDATA, I D=GAERTNER

INITIAL CATALOG

RP= 090 DAYS

CT ID= GAERTNER PFN=CDATA

CT CY= 001 0000256 WORDS.:

COMMAND-

Figure 3.1.2.b.4-3. Merging two PSL data files from two different PSL libraries into a single file as on a Type III system.

Program Requirements for Necessary New Support Software:

Paragraph 3.1.2.b.3 states a requirement which at this time is not fulfilled outside of a few experimental or research computer installations. However, the software which would satisfy this requirement is not terribly complex. It would require the following:

Input: a) The program (calling routine) for which the user wishes to generate the stubs.

b) File directories of existing PSL libraries.

Parameters: The language of the calling routine.

Program Performance:

- 1) Scan program, determining names of all routines called.
- 2) Scan library directories to determine which routines have already been created (i.e., which have listings in the PSL directory).

Note: Items one and two above describe what is usually a link editor function.

- 3) For all routines which are not yet written, generate stubs which contain the following:
 - a) Entry point.
 - b) Statements which cause printout of stub name in form "STUB name ENTERED".
 - c) Return point.

Element b) is considered necessary for two reasons: To facilitate debug and to catch typographical errors of names of called routines.

- 4) Flag in some manner each stub so created to mark it as automatically generated.

Output:

Program stubs which are in accord with specifications 3) and 4) above. They will be written in the language of the calling routine. The users could edit them if so desired.

An additional routine might also be written which would delete any or all program stubs which have been automatically generated, though the system's erase function would be relied on for this purpose as well.

3.1.3 Summary of Facilities for Source Data Maintenance

The key feature to easy source data maintenance is a high quality editor. Batch editing utilities, while functional, are awkward. Interactive editors offer by far the best means of system maintenance; the broader the scope and flexibility of the editor, the nearer optimal the system for PSL maintenance.

A summary of functional capabilities of these representative editors was given in the Existing Tools section of Paragraph No. SPS V-3.1.1.d. The discussions there and throughout Chapter 3 show that the only system which already complies with all standards is the Type II, or commercial time-sharing system. The sample Type I system requires some additional software in order to meet PSL requirements, but that software is readily implemented.

Only the Type III system falls far short of meeting the requirements. In order to bring that type of computer facility up to standard, a contractor either would need much additional software and perform extensive system modifications or would have to upgrade the operating system and remaining system software available for the given hardware. This latter approach would perhaps be the least costly on a long term basis. In terms of efficient software development and maintenance, it would be best to avoid a Type III system if at all possible.

3.2 Output Processing

Definition of Functional Division

The functional area concerns the output, i.e., listing or copying, of both library file contents and statistical data concerning library file storage. The Basic requirements involve only the output of printed listings, whereas the Full requirements also include additional output media and information.

3.2.1 Basic Output Processing Requirements

3.2.1.a Library Control Listings

Classification: Basic

Requirement:

Library control listings - A basic PSL implementation must also include the means to generate printed reports of a library's status. Such reports should contain the following:

1. Information related to the physical storage of files. This will include file storage space allocated and used, directory space allocated and used, type and serial number of storage device, data identification, and similar information.
2. A directory listing (i.e., an alphabetical list of file names) of the contents of a single library.

Purpose of Paragraph:

To define the library control listings required in PSL implementation.

Existing Tools Satisfying This Requirement:

Every known operating system which includes support for disk storage also includes a means of reporting that storage, and therefore would comply with the majority of the requirements expressed in this paragraph.

Examples:

Example 3.2.1.a.1: Generation of library control listings in a Type I system.

Most in-house minicomputer systems offer some form of file storage reporting. DEC's PIP allows the user to obtain reports with varying levels of detail. The most complete report possible results from applying both the /E and /W "switches" (i.e., instructions) to all files on a given device. This generates an unalphabetized directory listing of all files (file name and extension) and all unused areas, their sizes in blocks, creation dates, the absolute starting block (in octal), and the contents of each extra word per directory entry. The absolute starting block and contents of extra words is applicable to disk only.

In Figure 3.2.1.a-1 below, the extended listing is requested and stored as the file BX.LST. Next the user requests the listing of a non-existent file in order to obtain information on the number of free blocks. Since an alphabetized listing is desired, the sort utility is called and the directory list is alphabetized by filename and extension designation. This sorted listing may then be printed out to the console (TT) when PIP is re-entered. No device was specified in the initial listing request so the default device (DK, i.e., user's disk) is assumed.

This sample system requires the user to specify the type of storage unit for which he seeks information, and since this is echoed it is contained on the report. Serial numbers are not assigned to devices other than as part of the name, while data type is normally indicated by the extension designation. Therefore, all PSL requirements are fulfilled except for that concerning the listing of directory space allocated and used. Since directory space is allocated by the user upon disk initialization, the amount of space is not predetermined. However, it would not be difficult to modify the existing software to include directory reporting as well.

```

•R PIP
*BK.LST=***/E/W
*DU44Y.FIL/L
22-MAR-79
2671 FREE BLOCKS
*IC

•R BSORT

INPUT FILE?
BK.LST

OUTPUT FILE?
BK.SRT

*ENTER SORT FIELD DEFINITIONS
1 10

ENTER RECORD LENGTH
40

STOP --

•R PIP
*TT:=BK.SRT
103 FILES, 2077 BLOCKS
22-MAR-79
2671 FREE BLOCKS
30301V.BIN 13 4-MAY-73 1673
30301V.EDT 21 4-MAY-73 1710
< UNUSED > 1304
< UNUSED > 63
< UNUSED > 1304
AB .SAV 42 4-MAY-73 224
AS4030.OPC 4 4-MAY-73 1224
BA .SYS 6 19-MAR-79 2462
BATC1 .SAV 25 19-MAR-79 2470
.
.
.
T .LOG 64 21-MAR-79 3052
TEST .DAT 56 21-MAR-79 3474
TT .SYS 2 4-MAY-73 166
*IC

```

Figure 3.2.1.a-1. Generation of Library control listings in a Type I system.

Example 3.2.1.a.2: Generation of library control listings on a Type II system.

The majority of commercial time-sharing systems will supply storage data upon user issuance of a single command. In CSS, this command is STAT, and takes as parameter the filemode designation of the volume of interest or an "*" if all attached volumes are to be described. Note that multiple volumes (either disk segments or magnetic tapes) may be attached to a user concurrently. In the sample dialogue of Figure 3.2.1.a-2 the parameter has been omitted, thus information concerns the P disk, the user's permanent disk designation. The information supplied is the storage unit designation, storage type and access level (writable or read only), count of records used and those remaining, total number of records allocated, and the overall percent of use of the given total storage space. Data identification may be found by looking at the filetype on the directory listing. Directory storage information is not given by this command, nor by any command available in this system. In fact, none of the known time-sharing systems offer such information, though additional data may be given on disk storage itself. For example, CMS's QUERY DISK command gives the same information as the CSS STAT command, plus a statement of the number of files residing on the disk and the disk's virtual address.

Directory listings may also be readily obtained. In CMS, the LISTFILE command produces the directory listing of any or all disks. This will be in alphabetical order if the user has not stored any new files on the disk since logging on. The CSS System requires that the listing be explicitly alphabetized. However, Figure 3.2.1.a-2 shows, commands can be combined to perform a complex function which is user created. In this instance the LISTF, SORT and STAT commands were combined to form the DISKE user-built command to alphabetically list a directory.

EXAMPL60 MEMO P ----- 20APR79 ----- PAGE 1

12.18.23 >printf diske exec

```
&TYPE OFF
LISTF $1 $2 $3 (EXEC)
&STACK 17 24 0 15 27 28
SORT LISTF EXEC (BRIEF)
&BEGSTACK
  &S
    ZONE 1 6
    CHANGE //      / * *
    TOP
    ZONE 1 72
    CHANGE /        // * 1
    TOP
    I FILENAME FILETYPE MODE    ITEMS
    FILE
  &ENDSTACK
&STACK RT
EDIC .LISTF EXEC
&STACK RT
&PRINT ***DIRECTORY LISTING FOR PERMANENT DISK
&PRINT
STAT P
PRINTF .LISTF EXEC
ERASE .LISTF EXEC
ERASE LISTF EXEC
```

12.19.06 >diske

***DIRECTORY LISTING FOR PERMANENT DISK

F DISK(MT): 000081 USED, 000219 LEFT (OF 000300), 27% (OF 0002 CYL) ADAMS

FILENAME	FILETYPE	MODE	ITEMS
ALL	DATA	P	7
FIRST	DATA	P	3
SECOND	DATA	P	4
DISKE	EXEC	P	25
FIND	EXEC	P	14
PROFILE	EXEC	P	4
PX	EXEC	P	10
SCAN	EXEC	P	10
PROG1	FORTRAN	P	4
SAMPLE	FORTRAN	P	4
ADDRESS	MEMO	P	4
EXAMPL2	MEMO	P	55
EXAMPL23	MEMO	P	30
EXAMPL29	MEMO	P	36

Figure 3.2.1.a-2. Generation of library control listings in a Type II system.

Example 3.2.1.a.3: Generation of a library control listing in a Type III system.

Most, though not all, of the library control listings requirements are met by the Type III AUDIT utility. This facility allows the user to request data concerning the storage of his files by specifying AUDIT and his user ID. The P parameter in the example below specifies that a partial listing (i.e., not the full listing including flag settings) is desired. The response is to give for each file the file name (and therefore data identification as that is typically part of the file name), cycle number (equivalent to file version), storage unit serial number and number of physical record units occupied, as well as other data concerning dates of file creation and usage.

This Type III facility does not allocate a specific amount of storage to a user, but rather lets all users share the mass storage device. Therefore, information about space allocated is not applicable, nor is storage unit type since it must necessarily be disk. Directory storage is not a reportable item.

The listing in Figure 3.2.1.a-3 shows the files in non-alphabetical order. However the alphabetical clause of the requirement could be met by applying the SORT utility to the output of the AUDIT utility.

Specifications for Necessary New Support Software:

None of the systems examined offered information regarding the directory space allocated and used. A varying amount of effort would be required to meet this standard, depending on the type of system. The approach needed to make the change would also be dependent upon the type of system. For the Type III system the value of such reporting is dubious.

COMMAND- AUDIT, ID=ADEXO, AI=P

AUDIT OF 6000 PERMANENT FILES PARTIAL ID
16.08.16 07/08/76 PAGE NO. 1 TIME

SETNAME=SYSTEM:

OWNER	PRUS	PERMANENT FILE NAME	LAST ATT	CYCLE	ACCOUNT
ADEXO	2	DEMOPARTS	07/08/76	1	MGR333
0050		DEMOSRTD	07/08/76	1	MGR333
ADEXO	3	DIAGNOSTICBINNY	07/08/76	1	MGR333
0050		DIAGNOSTIC	07/08/76	1	MGR333
ADEXO	7	FOUND	07/08/76	1	MGR333
0052		DEMOSUMD	07/08/76	1	MGR333
ADEXO	2	DEMOSABS	07/08/76	1	MGR333
0051		USERDIAGNOSTIC	07/08/76	1	MGR333
ADEXO	5	07/08/76 10/06/76	07/08/76	1	MGR333
0052		07/08/76 10/06/76	07/08/76	1	MGR333
ADEXO	4	07/08/76 10/06/76	07/08/76	1	MGR333
0051		07/08/76 10/06/76	07/08/76	1	MGR333

AUDIT FINISHED
EXIT
COMMAND-

Figure 3.2.1.a-3. Library control listing generated on a Type III system.

3.2.1.b Source Data Listings

Classification: Basic

Requirement:

Source data listings - The computer installation must be able to provide printed listings of the data stored in a PSL. Such printout shall include:

1. Listings of selected data records.
2. Listings of all data in a single library data file.

Purpose of Paragraph:

To define the type of data listings required in a PSL implementation.

Existing Tools Which Satisfy This Requirement:

Some systems offer standard utility programs for printing data files (or portions), others contain a specific command for that purpose. Whatever the system, the editor can often provide the means for implementing this requirement.

Examples:

Example 3.2.1.b.1: Generation of printed data listings in a Type I system.

Printed listings may be obtained in several ways depending upon the hardware configuration of the system. If the user's terminal is a hard copy device, such as a teletype, then a printed listing is most easily obtained by entering the Editor and listing the line(s) of interest as illustrated in Figure 3.2.1.b-1 below. This would satisfy both classes 1 and 2 of the specification.

```
• R E
• ERSAMPLE.DATSRSS
• 2AS3LSS
  66.969  98.796  90.053  51.151  96.434
  18.246  41.567  85.188  37.022  55.437
  99.429  97.641  90.982  67.123  83.901
• 15A$2LSS
  2.221  37.637   5.838  96.291  25.208
  84.626  80.885  23.679  14.110  71.548
• EXSS
```

Figure 3.2.1.b-1. Listing of selected records of data using a Type I system.

However, if the user's terminal is a CRT then hard copy printout must be obtained using PIP. For the listing of an entire file this is a simple procedure, as is shown by Fig. 3.2.1.b-2. The file is simply copied onto the hard copy device that has been assigned the designation "LP", the line printer.

If it is desired to print only a portion of the file when the user's terminal is a CRT, the process becomes more complex. First, a new file must be created consisting of those lines which the user wishes to have printed. This is done by editing the original file and saving the lines of interest, then editing a new (output) file and issuing the UNSAVE command. Once the desired output has been placed in a separate file, PIP is accessed to print it. The sample dialogue below causes the sixth through eighth records of SAMPLE.DAT (as shown in Figure 3.2.1.b-2 to be printed using this technique).

```

•R EDIT
*ERSAMPLE.DAT$F$5A$3$B$/D$
*E/PATL.DAT$U$B$/L$
  66.767  73.776  70.053  51.151  76.434
  13.146  41.567  35.433  37.022  55.437
  77.423  77.641  70.732  67.123  33.701
*E<$
•R PIP
*LP:=PARTL.DAT
*
```

Figure 3.2.1.b-2. Listing selected records of data on a non-console device in a Type I system.

Example 3.2.1.b.2: Generation of printed data listings using a Type II system.

The commercial time sharing system offers the easiest solution to implementation of specific PSL requirements. Both systems studied in depth have a single command (PRINTF and PRINT in CSS and CMS respectively) which allows the user to print all of a file or any portion thereof. The CMS command routes printout to the printer while the CSS command generates output at the user's terminal. Figure 3.2.1.b-3 illustrates a request to print print lines 3 through 5 of SAMPLE DATA. If the entire file were to be printed, the parameters following the file specification would be omitted, letting the system assume the default parameter settings of first line and last line as the starting and ending points for printout. Printout of the entire file may also be obtained on a user designated printer with the OFFLINE PRINT command.

11.34.26 >PRINTF SAMPLE DATA 3 5

1.010	3.342	10.959	35.678	15.439
16.659	30.709	34.320	29.542	68.369
44.335	50.686	5.103	74.447	0.753

11.35.19 >

Figure 3.2.1.b-3. Listing of selected records of data by a Type II system.

If the user's terminal is not a hard copy device, printout of a portion of the file must again be obtained by placing in a file all records to be printed, then having that file printed offline. This operation is far simpler in CSS than on the smaller Type I system, as Figure 3.2.1.b-4 illustrates. The user enters only four commands, as opposed to the 12 commands needed to implement the same function on the in-house minicomputer.

```

12.30.08 >edit partial data
NEW FILE.
INPUT:
>
EDIT:
>set sample data # 3 5
  44.335  50.686   5.103  74.447   0.753
>file

12.30.43 >offline print partial data

12.30.55 >

```

Figure 3.2.1.b-4. Listing selected records of data on a non-console device in a Type II system.

Example 3.2.1.b.3: Generation of printed data listings using a Type III system.

The particular Type III system which we use as representative allows printout of all or part of a permanent file, of course, depending on the hardware configuration of the particular facility. If the user has a hard copy terminal, the easiest method is to access the file, edit it, and list the desired lines, either by specific line number as in the example below or by specifying "L, A" if the whole file is to be printed. If the user is at a CRT terminal, he may use the PAGE command to create a print file which can then be output on the printer. This process is rather tedious.

COMMAND- ATTACH, S, SAMPLEDATA, I D-GAERTNER

PF CYCLE NO:..= 001

COMMAND- EDITOR

..E, S, S

..L, 120, 140

120=	71.531	90.235	97.627	73.649	63.250
130=	16.659	30.709	34.320	29.542	68.369
140=	44.335	50.686	5.103	74.447	0.753

..BYE

COMMAND-

Figure 3.2.1.b-3. Listing of selected records of data produced by a Type III system.

3.2.1.c Control Data

Classification: Basic

Requirement:

Control data - All printed listings produced by the library must contain the internal library and file names and the date and time produced for use in maintaining the external libraries.

Purpose of Paragraph:

To define additional information required on PSL listings.

Examples:

Example 3.2.1.c.1: Generation of additional data on Type I library listings.

- (a) Directory listings as shown in Figure 3.2.1.a-1 contain date of printout, name of files, and their respective libraries (in that libraries are designated by extension name). Time is not given; however, the TIME command could be issued prior to printout at the teletype. If printout is routed to the lineprinter this can't be done. The only other alternative is to modify the PIP listing command to include the time.
- (b) Files listed in the editor (see Figure 3.2.1.b-1) will contain the file name and library name in the edit request, but will not include the date or time. If the user issues the DATE and TIME commands prior to entering the editor, the listing would meet the standard. Those commands would also be necessary to fulfill requirements when file listings are done with PIP on the teletype. However, those commands do not display on the line printer, therefore PIP listings (see Figure 3.2.1.b-2) on that device cannot conform to standard unless PIP itself is altered or additional line printer commands added to the system.
- (c) Compiler listings, as shown in Figure 3.2.1.c-1, contain the data and time but fail to give the file and library names. To bring this system up to standard, it would be necessary to have the compiler call a print routine which would list the desired information.

```

:IT-11 FORTRAV IV          V01B-03      THU 22-MAR-73 05:30:52      PAGE 001
      C      SAMPLE PROGRAM
      C
0001      READ(5,1000) V,M
0002 1000      FORMAT(2I4)
0003      L = V**M
      C
0004      WRITE(6,1001) V,M,L
0005 1001      FORMAT(1X,I4,4I ** ,I4,3I = ,I10)
0006      STOP
0007      END

```

```

:IT-11 FORTRAV IV          STORAGE MAP
NAME      OFFSET  ATTRIBUTES
V          000042  INTEGER*2 VARIABLE
M          000044  INTEGER*2 VARIABLE
L          000046  INTEGER*2 VARIABLE

```

Figure 3.2.1.c-1. Compiler listing on Type I system.

Example 3.2.1.c.2: Generation of additional data on Type II listings.

- (a) Directory listings in this system contain the time, file name and library name (i.e., file type, the second part of the file designation), but not the date. This is easy to alter, as the user built command DISKE (reference Figure 3.2.1.a-2) could simply be altered to include the date command.
- (b) File listings at the terminal also include all needed items except for the date. Since the date may be listed via the DATE command, this lack poses no problem. File listings that result from the OFFLINE print command contain all required data (see Figure 3.1.1.d.5.c-1).
- (c) Compiler listings in this system contain all needed information, as is shown below (Figure 3.2.1.c-2).

EXAMPL66 MEMO P ----- 20APR79 ----- PAGE 1

LEVEL 2.2.1 (DEC 77)

OS/360 FORTRAN H EXTENDED

FILE: PROG1 VP/CSS --- NATIONAL CSS, INC. (STAMFORD DATA CE
REQUESTED OPTIONS: SIZE=337K,TERM
OPTIONS IN EFFECT: NAME(MAIN) NOOPTIMIZE LINECOUNT(55) SIZE(0337K) AUTODBL
SOURCE EBCDIC NOLIST DECK NOOBJECT NOMAP NOFORMAT NOGOS

C SAMPLE PROGRAM

C

ISN 0002 READ(5,1000) N,M

ISN 0003 1000 FORMAT(2I4)

ISN 0004 L = N ** M

C

ISN 0005 WRITE(6,1001) N,M,L

ISN 0006 1001 FORMAT(' ',I4,' ** ',I4,' = ',I10)

ISN 0007 STOP

ISN 0008 END

*OPTIONS IN EFFECT*NAME(MAIN) NOOPTIMIZE LINECOUNT(55) SIZE(0337K) AUTODBL

*OPTIONS IN EFFECT*SOURCE EBCDIC NOLIST DECK NOOBJECT NOMAP NOFORMAT NOGOS

STATISTICS SOURCE STATEMENTS = 7, PROGRAM SIZE = 346, SUB

STATISTICS NO DIAGNOSTICS GENERATED

***** END OF COMPILATION *****

232

Figure 3.2.1.c-2. Compiler listing in a Type II system.

XTENDED DATE 79.076/13.36.08 PAGE 1
ID DATA CENTER) HSYS SATURDAY 17 MARCH 1979

() AUTODBL(NONE)
RMAT NOGOSTMT NOXREF NOALC NOANSF TERM FLAG=I

PR000010
PR000020
PR000030
PR000040
PR000050
PR000060
PR000070
PR000080
PR000090
PR000100

K) AUTODBL(NONE)
RMAT NOGOSTMT NOXREF NOALC NOANSF TERM FLAG=I
346, SUBPROGRAM NAME = MAIN

232K BYTES OF CORE NOT USED

Example 3.2.1.c.3: Generation of additional data on a Type III listing.

- (a) Directory listings obtained via the AUDIT command (see Figure 3.1.2.a-3) contain the date, time and names of the files. However, since this particular system does not support a compound file designation (e.g., filename, filetype or library name, filemode or location), the library name is not included unless the user has made it part of the file name.
- (b) File listings as produced at a hard copy terminal via the editor (Figure 3.2.1.b-3) contain the file name in the edit request, but neither date nor time. The user may cause printout of these by using the CLOCK and date functions before entering the editor.

The listing produced with the PAGE command contains the time of the listing but does not contain the remaining required information. Additional software would be needed to cause time and date to print out.

- (c) Compiler listings in this representative system, like the Type I system, contain the date and time but not the file designation. (See Figure 3.2.1.c-3). Once again, additional software would be required to bring this system up to standard.

PROGRAM BINNY		73/74	OPT=1	FTN 4.6+4
1		PROGRAM BINNY		
		1 (INPUT=260, OUTPUT=260, TAPE4=260, TAPE8=260, TAPE1=260, TAPE3=260, TAPE9=260, TAPE6=OUTPUT, TAPE5=INPUT, TAPE30=260, TAPE7=260)		
5	C			
	C	W.W.GAERTNER RESEARCH INC. BINSRCH MAIN MAY3074		
	C	205 SADDLE HILL ROAD		
	C	STAMFORD, CONNECTICUT 05903		
	C	TELE (203) 322-7661		
10	C			
	C	COMPANY PRIVATE		
	C	AUTHOR: W.M.SCHREYER		
15	C	THIS PROGRAM SEARCHES THE DATABASE FOR THE INPUT PARTS		
	C			
	C	INPUTS		
	C	FI 1 DSK INPUT SORTED		
	C	FI 8 TAP2 DATABASE		
20	C			
	C	OUTPUTS		
	C	FI 2 DSK OUTPUT DATABASE		
	C	FI 3 DSK PRIMARY INPUT NOT FOUND		
	C	FI 4 DSK FINAL NOT INDB		
25	C	FI 7 DSK OUTPUT DIAGNOST		
	C	FI 9 DSK PARTSIN FOUND		
	C			
		IMPLICIT INTEGER(A-Z)		
		REAL FLOAT, PCNOTF		
30	C	DATA DEFINITIONS		
	C			

Figure 3.2.1.c-3. Compiler listing in a Type III system.

FTN 4.6+420

01/20/77 13.43.38

PAGE 1

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

U, TAPE1=250, TAPE2=250
INPUT

MAIN MAY3074

THE INPUT PARTS LIST

3.2.2 Full Output Processing Requirements

Classification: Full

Requirement:

Full Output requirements include all of the Basic Output requirements and the following additional requirements.

Purpose of Paragraph:

To define which paragraphs constitute full implementation of Output Processing Requirements.

3.2.2.a Full Magnetic Tape Output

Classification: Full

Requirement:

Magnetic tape output - The Full implementation of PSL output processing must include the capability to copy data onto magnetic tape for permanent storage or for distribution to other computer facilities. The user must be able to determine which data is to be copied, i.e., access said data, in at least one of the following two ways:

1. Copy one or more specific data records onto magnetic tape.
2. Copy one or more specific data files onto magnetic tape.

Purpose of Paragraph:

To specify the need for magnetic tape copying capabilities and to define the units by which data may be accessed for such copying.

Examples:

The three representative systems used here all allowed magnetic tape output of individual files, thus satisfying this requirement. There are a few systems which are record-oriented rather than file-oriented in magnetic tape data transmission, but the process by which these work does not differ appreciably from file-oriented transmission.

Example 3.2.2.a.1: Output to magnetic tape on a Type I system.

The standard DEC system utility, PIP, performs disk-to-tape data transfer easily. As shown in Figure 3.2.2.a-1, after calling PIP the user simply requests that the file SAMPLE.DAT (which resides on disk DK, the default device designation) should be copied to MT0 (magnetic tape unit 0). The save file name is retained. As verification, the contents of the volume mounted on device MT0 may be listed.

```
.R PIP
*MT0: SAMPLE.DAT= SAMPLE.DAT
*MT0: /L
22-NOV-76
SAMPLE.DAT      1 22-NOV-76
*
```

Figure 3.2.2.a-1. Copying one file of data onto magnetic tape using a Type I system.

Example 3.2.2.a.2: Output to magnetic tape on a Type II system.

The majority of commercial time-sharing systems offer utilities to perform disk-to-tape and tape-to-disk data transfer. CMS offers TAPE and TAPPDS commands. CSS offers OFFLINE, TAPE, TAPIO, and UTILITY commands. The latter is the most powerful and will be the one used in the example in conjunction with the TAPE command (see Figure 3.2.2.a-2).

In a time-share system it is necessary to notify the operator to mount the tape; such request is not necessary in a Type I system because the user performs such tasks. Once the tape drive is attached and the tape is mounted, the program UTILITY is requested. Note that the TAPE REWIND command was issued first to ensure that the tape had actually been mounted.

When UTILITY is requested without parameters, it automatically goes into conversational mode and prompts the user, who then enters the needed information concerning the tape and the data to be transferred. In dialogue below, the output tape and its density is defined, then the input is specified as originating on user's disk (DT). The file name is then entered along with the blocking format, after which the transfer occurs and the next operation is prompted. This might be tape verification, transfer of additional files, or exit from the program as shown below. This was the only type of system reviewed which allowed transfer of individual records to tape.

12.45.08 >MOUNT TAPE X3476 AS 282 RINGIN 9TRK-LO

12.46.12 >DEV 282 ATTACHED
>T REW

12.51.35 >UTILITY
REQUEST: OUT=TAPE2
REQUEST: DEN,OUT,2
REQUEST: DT
INPUT FILE?
SAMPLE DATA
BLOCKING FACTOR
1
REQUEST: END

12.52.16 >

Figure 3.2.2.a-2. Copying one file of data onto magnetic tape using a Type II system.

Example 3.2.2.a.3: Output to magnetic tape on a Type III system.

This system does not offer the usual interactive copying utilities, but the standard can be met by processing a user-written copy program in batch mode. Figure 3.2.2.a-3 shows the sample job stream, including the source code of the FORTRAN copying program, which copies the file SAMPLEDATA onto TAPE2. This job may be submitted to the control site from the remote site assuming that the remote site includes a card reader and correct type of user terminal.

```

COPYFIL,NT1.
COMMENT.(AAA-BBB,NNNNNT),JONES
ATTACH,TAPE1,SAMPLEDATA,ID=GAERTNER.
REQUEST,TAPE2,NT,S. BIN/1122,REEL/4031, RINGIN.
FTN.
LGO.
*EOR
      PROGRAM CPYFL(TAPE1,TAPE2,OUTPUT)
      REAL A(8)
      FORMAT(8A10)
2      READ(1,2)A
1      IF(EOF(1).NE.0) STOP
      BUFFER OUT (2,0) (A(1),A(8))
      WAIT=UNIT(2)
      GOTO 1
      END

```

Figure 3.2.2.a-3. Copying one file of data onto magnetic tape using a Type III system.

General Observations on Magnetic Tape Output

The use of magnetic tape as a means of data transfer between computer facilities is highly reliable, as shown in the March, 1976 DATAMATION article, "Erasing Myths about Magnetic Media". Unfortunately tapes are written in a variety of formats and codes at different installations. For that reason it is considered necessary to outline the code conversion facilities available in the three representative computer systems.

Type I - Text files are handled as ASCII. However, PIP allows copying in image mode, which would facilitate code conversion processing by user-written programs on either input or output files. The existing utilities can handle only DEC formatted materials.

Type II - This system supports both ASCII and EBCDIC input terminals, and is therefore geared to handle both types of encoding. The data transfer program UTILITY includes a parameter which specifies the code type (ASCII or EBCDIC), and there exists a system command (EBCD) to translate files from BCD to EBCDIC. This flexibility allows the system to process materials formatted by IBM, DEC, CDC, and other major distributors.

Type III - Although this system is BCD oriented, it does support facilities to handle input and output of tapes in ASCII and EBCDIC as well. This allows processing of magnetic tape materials from most major distributors.

3.2.2.b Full Punched Card Output

Classification: Full

Requirement:

Punched card output - The Full implementation of PSL output processing must include the capability to output data on punched cards. The user must be able to select data for output in terms of either records or files, i.e., he must be able to use at least one of the following methods:

1. Punch the contents of one or more data records.
2. Punch the contents of one or more data files.

Purpose of Paragraph:

To specify the need for punch card output capabilities and to define the units by which data may be accessed for such output.

Examples:

Example 3.2.2.b.1: Punched card output in a Type I system.

Although card-punches are not usual peripheral equipment in a Type I system, the software is often designed to support such contingencies. DEC's RT-11 system establishes physical device names at time of system generation. If a card punch has been included in the generation, its usual designation is CP. Card punch output is then obtained via PIP much as was magnetic tape output. The sample dialogue is shown below in Figure 3.2.2.b-1.

```
•R PIP
*CP: SAMPLE. DAT= SAMPLE. DAT
*! C
```

Figure 3.2.2.b-1. Punching a file to cards on a Type I system.

Example 3.2.2.b.2: Punched card output in a Type II system.

Punch card output is typically readily obtainable in a variety of manners. CMS has the PUNCH command as well as the SPOOL command to modify options for punch spool devices. CSS's UTILITY may be used to punch cards either from disk or from magnetic tape. Alternately, if the file is disk resident, the OFFLINE PUNCH command may be used as shown below (see Figure 3.2.2.b-2).

13.08.41 >OFFLINE PUNCH SAMPLE DATA

13.08.59 >

Figure 3.2.2.b-2. Punching a file to cards on a Type II system.

Example 3.2.2.b.3: Punch card output using a Type III system.

Unlike the preceding systems, the Type III installation does not allow logical or virtual device assignment. Files may only be punched by sending them to the control processing site with a PUNCH request, as shown in Figure 3.2.2.b-3. The file must be a local file, either newly created or an attached permanent file, before it can be transmitted for punching. In the sample dialogue the user first enters the editor only for file verification.

COMMAND- ATTACH, S, B55SAMPLEDATA, ID=GAERTNER

PF CYCLE NO. = 001

COMMAND- EDITOR

..E, S, S

..L, A, S

0.381	1.657	6.509	24.143	86.271
0.341	25.607	50.576	72.991	82.768
39.686	93.204	2.049	73.456	22.296
72.675	35.383	58.224	30.897	61.368
0.381	1.657	6.509	24.143	86.271
0.341	25.607	50.576	72.991	82.768
39.686	93.204	2.049	73.456	22.296
72.675	35.383	58.224	30.897	61.368

..BYE

COMMAND- BATCH, S, PUNCH, GAER

FILE NAME-IGAERA0 ,DISP-PUNCH ,ID-**
COMMAND-

Figure 3.2.2.b-3. Punching a file to cards on a Type III system.

3.2.2.c Full Management Control Listings

Classification: Full

Requirement:

Full Management Control Listings - Additional printed output requirements are defined in the Management Data Collection and Reporting requirements.

Purpose of Paragraph:

To refer the reader to that section of the report which describes requirements for printed output of Management Data.

3.2.2.d Full File Directory and Listings by Programmer

Classification: Full

Requirement:

Full File directory and listings by programmer - A full PSL implementation must include the capability to generate a list of all files and/or listings thereof, that are identified as having been generated by a specific programmer.

Purpose of Paragraph:

To specify the need for a list and/or listings of all programs and data for which a programmer is individually responsible.

Examples:

Example 3.2.2.d.1: Listing of files by programmer in a Type I system.

File designations in this system consist of a six character file name and three character extension name. Traditionally, per DEC software, the extension name is indicative of the type of file and/or the library to which it belongs, e.g., .OBJ for object module. The filename may also be made indicative of the programmer, project and purpose of the file by using a coding system. In this manner the small in-house system can be made to comply with the requirements of this paragraph, even though it does not usually contain any software which facilitates such compliance.

One small contractor has written a utility program that uses filename standardization to produce file lists according to programmer and/or project code. The operation of this utility, called DIR, is shown in Figure 3.2.2.d-1. After being called, the user requests a list of all data files written by B (programmer code) for project 55 (B55.DAT). System response is the file name and extension, number of blocks, and date of creation of all files fitting those specifications. In order to obtain listings of those files, the user then may enter PIP and request printout in the manner previously discussed. Although DIR is installation specific, it is conceptually simple and could easily be implemented by individual contractors at little cost; or, a general version of it would be supplied by the Government as part of the PSL. The program specifications are given in the section entitled "Requirements for Necessary New Support Software" associated with this paragraph.

.R DIR

**DIR
B55.DAT**

B55MIN.DAT	1	19-NOV-76
B55MAX.DAT	1	19-NOV-76
B55AVG.DAT	1	19-NOV-76

**DIR
'C**

.R PIP
***TT:=B55MIN.DAT, B55AVG.DAT, B55MAX.DAT**
1.00 -7.00 63.00
38.00 -33.00 118.00
73.00 -1.00 176.00

Figure 3.2.2.d-1. Programmer directory and file listings generated on a small contractor's Type I system.

Example 3.2.2.d.2: Listing of files by programmer in a Type II system.

Both of the Type II systems examined in depth have file designations consisting of an eight character filename, eight character filetype, and one alphabetic character filemode (disk designation). The file type usually designates the library to which the file belongs, e.g., FORTRAN for Fortran source code. As in the preceding example, the file name can be used to designate the programmer, project and purpose of the file. Since the LISTF system command allows embedded, leading or trailing asterisks in file name and file type designation, it is possible to obtain listings of files by programmer, by project, by purpose, or by any combination of those qualifications. For example, to list all files created by programmer B which were involved with the generation of random numbers (RANDU), the request would be " LISTF B* RANDU *".

The example of Figure 3.2.2.d-2 demonstrates the listing of all data files created by B for project 55. The E option creates an EXEC file which may then be used in conjunction with the PRINTF command to list the contents of all the files of interest.

13.29.34 >LISTF B55* DATA (E NOITEM)

13.29.57 >P LISTF EXEC

41	42	B55RANDU	DATA	P
41	42	B55MAXIM	DATA	P
41	42	B55MINIM	DATA	P

13.31.12 >LISTF PRINTF

13.31.38 PRINTF B55RANDU DATA P

29.014 57.732 31.001 7.887

13.32.03 PRINTF B55MAXIM DATA P

100.000 100.000 100.000 100.000

13.32.54 PRINTF B55MINIM DATA P

1.000 1.000 1.000 1.000

13.33.21 >

Figure 3.2.2.d-2. Programmer directory and file listings generated on a Type II system.

Example 3.2.2.d.3: Listing of files by programmer in a Type III system.

The representative system utilizes 40 character file designations. This allows the user to construct compound file names which indicate programmer name, project, and program purpose. Unfortunately, the AUDIT utility (see Example 3.1.2.a.3) does not allow partial specification of file name, unlike the Type II system; nor is there any means of constructing a user program that would interface with the AUDIT utility as was shown for the Type I installation. However, the results of AUDIT may be placed in a local file which may then be edited as shown in the following sample dialogue (Figure 3.2.2.d-3). Note that the LIST command allows the user to have listed only those records which contain a specified character string, thus limiting items listed to those files of interest. These files may then be attached and printed via BATCH PRINT or PAGE.

COMMAND- AUDIT,LFN=LIST,AI=P,ID=GAERTNER

AUDIT FINISHED

EXIT

COMMAND- EDITOR

..E,LIST,S

..L,/B55/,A,(15,18)

130=	GAERTNER	B55MAXIMUMDATA					1
MGX111	0050	2	07/08/76	10/06/76	11/22/76	10/22/76	
170=	GAERTNER	B55MINIMUMDATA					1
MGX111	0050	2	07/08/76	10/06/76	11/22/76	10/22/76	
260=	GAERTNER	B55AVERAGEDATA					1
MGX111	0050	2	07/08/76	10/06/76	11/22/76	10/15/76	

..BYE

COMMAND- ATTACH,X,B55MINIMUMDATA,ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- BATCH,X,PRINT,GAER

FILE NAME-IGAER17,DISP-PRINT,ID=**

COMMAND- ATTACH,X1,B55AVERAGEDATA,ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- BATCH,X1,PRINT,GAER

FILE NAME-IGAER21,DISP-PRINT,ID=**

COMMAND- ATTACH,X2,B55MAXIMUMDATA,ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- BATCH,X2,PRINT,GAER

FILE NAME-IGAER22,DISP-PRINT,ID=**

COMMAND-

Figure 3.2.2.d-3. Programmer directory and file listings generated on a Type III system.

Requirements for Necessary New Support Software:

The additional software required by a Type I system has the following specifications:

Input:

From user: File name and extension of interest.

From system: Total directory listing of attached devices.

Function:

Polling devices, i.e., for each device directory,

- 1) Create temp file of directory
- 2) Place device designation in output file
- 3) Scan user input filename and extension, and parse into allowable combinations for matching. E.g., if input were "B55.DAT" and the universal match character were @, then this parser would produce the string "B55@@@.DAT". If input string were "*B5*.DAT" the parser would produce strings:

B5@@@@.DAT
@B5@@@.DAT
@@B5@@.DAT
@@@B5@.DAT
@@@@B5.DAT

- 4) For each parser string, scan device directory for match. If match, place directory entry in output file.

Output: A listing of all files on all attached directories that meet the input specifications.

3.2.2.e Full Automatic Indentation of Source Code Listings

Classification: Full

Requirement:

Full automatic indentation of source code listings - The Full implementation of a PSL must include the capability to automatically indent source code listings generated from PSL source data files. Indentation for JOVIAL, ANS FORTRAN, ANS COBOL, and TACPOL is in accordance with the structured programming standards defined in SPS Volume I, "Programming Language Standards".

Purpose of Paragraph:

To specify the need for automatic indentation of source code as an integral part of structured programming.

Existing Tools Which Satisfy this Requirement:

A precompiler would properly handle this function for those languages which require a precompiler. Languages such as PL/1 which support all structured programming figures (IFTHENELSE, DOWHILE, DOUNTIL, and CASE) also include an analysis printout of logical nesting levels. Even though these language compilers do not currently support automatic indentation, the nesting level monitor could be readily adapted to include that feature.

Requirements for Necessary New Support Software:

It is suggested that a routine for automatic indentation be developed and included by the Government as part of the standard PSL package for its contractors. As stated previously, such a routine could be callable by the precompiler or the compiler, depending upon the language. It is probable that different versions of the routine would be needed by the two calling sources.

The version which is callable by a compiler would use the parsing stack to determine the indentation level. It would then take the keyword grouping as passed by the compiler and store it in an output buffer with an indicator of its starting column.

The version to be called by the precompiler would also make use of the parsing stack. The problem with using the precompiler as a basis is that it would only parse those control structures which are key elements of structured programming but which are not included in the standard language. Thus not all control structures and statements would be parsed. As a result, the indentation routine would either have to implement its own parsing stack or would have to be designed as precompiler-dependent.

In either case, the indentation routine would be essentially table driven. The table would include the list of keyword groupings and flags indicating if a given grouping (a) should itself be indented, and (b) if it causes indentation in the next keyword grouping. For example, in the following FORTRAN source segment, the DO statement should not necessarily be indented, but it would cause the subsequent assignment statement to be indented.

```
      DO 1010 I=1,N
      IPER (I)=I
1010  CONTINUE
```

A further consideration for FORTRAN source code indentation would be the statement numbers and comment and continuation indicators which appear in the left margin. The actual source code should be indented, but the marginal material cannot be.

The basic logic for any indentation scheme would be as follows:

- 1) Determine source code type and set indicator to select appropriate table.
- 2) Read a segment of input file.
- 3) If end of file, then exit to calling routine.
- 4) Should that segment be indented and if so to the right or left. If it is to be indented then increment (decrement) indentation field pointer.
- 5) Set column indicators for output of code segment.
- 6) Place code segment in output buffer using proper column alignment.
- 7) Return to 2).

3.2.2.f Full Data Scanning Capability

Classification: Full

Requirement:

Full Data Scanning Capability - A Full implementation of a PSL must include the capability to scan for a specific string of data in every record of every file in a PSL and to list the names of the files in which it appears. The capability allows a user to determine where a specific string of data is used and if it can be replaced in every file in which it is used.

Purpose of Paragraph:

To specify the need for data scanning capability throughout the PSL.

Examples:

Example 3.2.2.f.1: Scanning PSL files for a specific string using a Type I system.

This system does not offer facilities for direct complementation of an automatic scanner. However, the function can be accomplished indirectly by individually editing the PSL files and scanning them for occurrences of the specified string. The Figure 3.2.2.f-1 demonstrates the use of such a procedure to locate the string COEFF in the files B55CMA.FOR, B55CMB.FOR, and B55MAT.FOR. The user must determine which files need to be edited, however, once the edit mode is entered a macro can be constructed which will quickly search for all string occurrences within those files.

```
• R E
• ERB55CMA.FOR$RSM/GCOEFFSVS/$99EMSS
  CODE =ALPHA * BETA *COEFF
  SAVCOF =COEFF
  COEFF =SAVCOF
  ?*SRQH FAIL IN MACRO*?
• B/KSS
• ERB55CMB.FOR$R$B$99EMSS
  ?*SRQH FAIL IN MACRO*?
• B/KSS
• ERB55MAT.FOR$R$B$99EMSS
  C          MATCH COEFFICIENT TO ARRAY
    IF(COEFF.EQ.ARRAY(I,J) ) GOTO 100
  C          COEFFICIENT MATCHED TO ARRAY
  ?*SRQH FAIL IN MACRO*?
• ! C
```

Figure 3.2.2.f-1. Scanning for a specific string of data in every line of every file in a PSL using a Type I system.

Example 3.2.2.f.2: Scanning PSL files for a specific string using a Type II system.

Here the facility to construct new commands from existing system tools is used to build a command for scanning. SCAN builds a list of files to be searched and then applies FIND to each file in the list. FIND enters the editor and locates every occurrence of the argument string.

Example 3.2.2.f.3: Scanning PSL files for a specific string using a Type III system.

The method used to implement this requirement on a Type III system is similar to that used in the Type I system. As Figure 3.2.2.f-3 shows each file is in turn attached and edited, then scanned for the occurrence of the specified string. Fortunately, files can be attached from inside the editor, but once again the determination of which files are to be processed must be made by the user. It cannot be done automatically.

COMMAND- ATTACH,X,B55COMPUTEAFORTRAN,I D=GAERTNER

PF CYCLE NO.= 00:

COMMAND- EDITOR

..EX,S

..L,/COEFF/,A

190= CODE =ALPHA *BETA *COEFF
380= SAVCOF=COEFF
1520= COEFF =SAVCOF

..ATTACH,XX,B55COMPUTBFORTRAN,I D=GAERTNER

PF CYCLE NO.= 001

..EXX,S

..L,/COEFF/,A

..ATTACH,XE,B55MATCHFORTRAN,I D=GAERTNER

PF CYCLE NO.= 001

..L,/COEFF/,A

350=C MATCH COEFFICIENT TO ARRAY
470= IF(COEFF .EQ.ARRAY(I,J)) GOTO 100
610=C COEFFICIENT MATCHED TO ARRAY

..

Figure 3.2.2.f-3. Scanning for a specific string of data in every line of every file in a PSL using a Type III system.

14.42.40 >printf scan exec

```
&TYPE OFF
&IF &INDEX LT 1 &GOTO -NOARGS
&ALPHA1 = / !! &1
&ALPHA1 = &ALPHA1 !! /
LISTF &2 &3 &4 (CSS NOHEADER NOITEM)
EXEC CSS EXEC FIND
ERASE CSS EXEC
&EXIT
-NOARGS &PRINT ?SPECIFY STRING AND [ FILE SPEC.]
```

14.42.46 >printf find exec

```
&TYPE OFF
&IF &INDEX LT 3 &EXIT
&ERROR &GOTO -NOFILE
STATE &1 &2 &3
&PRINT
&PRINT FILE: &1 &2 &3
&STACK SETBR $
&STACK X LOCATE &ALPHA1 $ LINENO $ PRINT
&STACK X 999999
&STACK QUIT
EDIT &1 &2 &3 SB
&EXIT
-NOFILE &ERROR CONTINUE
&PRINT ?FILE &1 &2 &3 NOT FOUND
```

14.44.53 >scan print * exec P

```
FILE: FIND EXEC P
5:
&PRINT
6:
&PRINT FILE: &1 &2 &3
8:
&STACK X LOCATE &ALPHA1 $ LINENO $ PRINT
14:
&PRINT ?FILE &1 &2 &3 NOT FOUND
```

FILE: CLEAR EXEC P

FILE: GET EXEC P

FILE: PX EXEC P

```
6:
PRINTF &FN MEMO P
```

FILE: SCAN EXEC P

```
9:
-NOARGS &PRINT ?SPECIFY STRING AND [ FILE SPEC.]
```

Figure 3.2.2.f-2. Scanning for a specific string of data in every record of every file in a PSL using a Type II system.

3.2.2.g Full Directory Availability Printout

Classification: Full

Requirement:

Full Directory Availability Printout - A Full implementation of a PSL must include the capability to generate a printed report of:

- a) the initial space allocation made for a directory on a directory oriented device,
- b) the date of such allocation,
- c) the number of directory records currently utilized and/or the present space use within the directory,
- d) the date and time of the report,
- e) the date of the last directory compressions.

Examples:

None of the three systems examined included facilities for printout of directory space allocation. In fact, by far the majority of existing systems contain information on initial directory allocation only in the reference manuals. In order to rectify this situation, it would be necessary for the Government to supply a directory status report program as part of the standard PSL package. Specifications for such software are contained in the following subsection entitled "Requirements for Necessary New Support Software".

Requirements for Necessary New Support Software

Since there is a wide variance in directory size and format in different devices, the directory status report program would require initialization information for each device. It would also need to access information from the system device tables.

Upon entering, the program would request the user to specify what physical unit his request concerned and if he wished a status report or an initialization procedure. The latter would be used when a new storage device was being initiated into the system.

a) Initialization procedure.

- 1) Program requests user to specify device designation (e.g., disk number), starting address of directory, and directory size in bytes.
- 2) Program obtains date and time from system.

b) Report Procedure

- 1) Program scans directory area, notes address of last used byte, calculates difference between that and end of allocated space, and tallies number of imbedded unused directory entries.
- 2) Program reports the following:
 - (a) Initial directory space allocation,
 - (b) Date of initialization,
 - (c) Remaining directory space in bytes,
 - (d) Number of directory bytes which can be retrieved via compression,
 - (e) Date of last directory compression.

3.2.3 Summary of Facilities for Output Processing

There is no one standard facility which is key to fulfilling PSL requirements for output processing. In some systems, special file-handling programs such as PIP meet this need. In others, e.g., CSS or CMS, file manipulation including output, is done with simple system commands. Older systems that were designed around primarily batch processors do not fully integrate the storage media other than cards into their command structure. The result is that reporting of system status and file status to a user (as opposed to an operator) is awkward at best.

Of the three representative types of computer installations, the Type II comes closest to allowing complete implementation of PSL requirements. The majority of additional software needed can be easily created by the user, as was shown in various examples. This flexibility is possible because of the so-called EXEC files which allow the user to build his own command functions from the basic system facilities.

The small in-house computer also allows fairly complete output processing, although the means for obtaining certain output is far more awkward than in the commercial time-sharing system. In some instances (e.g., labelling of printed output as to file and library names, date, and time), the Type I facility altogether fails to meet basic requirements. Fortunately, being a small system, it tends to be less complex than the other two, thus making integration of additional software a much easier task.

Such is unfortunately not true of the Type III installation, where output labelling also falls below standard. While additional software would be required to raise output processing capabilities to even Basic level, that new material could not be so easily meshed into the already-existing software system. Although it is possible, it would definitely be more costly to the contractor than if he initially had a different system.

3.3 Programming Language Support

Requirement:

Programming Language Support - This functional area involves the validation and compilation of program source code stored in the PSL. The basic requirements involve the compilation of source code. The full requirements involve the validation of source code based on structured programming standards.

Purpose of Paragraph:

To introduce the Programming Language Support functional area and to delimit the scope of the Basic and Full requirements within this area.

3.3.1 Basic Programming Language Support Requirements

3.3.1.a Compiler Interface

Classification: Basic

Requirement:

Compiler Interface - The Basic implementation of a PSL must include a method for invoking precompilers, if available, and compilers in order to process source statements stored in the library. Such source statement processing must include:

1. Syntax checking of source code (i.e., precompiler and compiler checking of source code for proper use of the language syntax and for identification of all errors).
2. Compilation of source code and storage of the resulting object module in the library.

Purpose of Paragraph:

To define the software tools required for compiler interface and to delineate their responsibilities.

Existing Tools which Satisfy this Requirement:

- 1) Precompilers - To date, there are only a few commercially available precompilers, and the majority of these are designed to facilitate structured programming in FORTRAN. IFTRAN has been available

since 1975, while the better known RATFOR (Rational Fortran) has been widely marketed in an even shorter span of time. Software Consulting Services started offering SFORTRAN in 1977, two years after they introduced SCOBOL, which is probably the best known of the COBOL precompilers.

The nature of both COBOL and FORTRAN is such that none of the structured programming key control structures can be directly implemented. Since these are the main high level languages currently used in production programming, most prospective Government contractors have only two means of meeting SP standards: Either purchase one of the linguistic preprocessors previously mentioned or implement the key control structures using methods such as those suggested in SPS Volume I, Section 4.

- 2) Compilers - Every major computer language and many of the minor ones have multiple compiler versions, the difference lying in tailoring for specific machines and in amount of support of new language constructs (e.g., one-in, one-out control structures such as DOWHILE and CASE). All compilers have the dual purpose of syntax checking and translation into machine code, though the quality of the syntax checking and space/time optimization of the resultant machine code may differ. It is unfortunate but true that the compilers which have been designed to translate the most sophisticated structured languages into code suitable for the newest most efficient machines simply are not used. There are few programmers who are familiar with the newer languages that have been developed in accordance with structured programming concepts. Moreover, it is impossible for all but heavily funded research institutions to take advantage of the latest breakthroughs in computer hardware technology. As a result, production computer installations use the older, more established compilers and equipment.

Not all of these compilers are designed to save the object modules, "compile and go" compilers are intended primarily for instructional environments. However, all those languages which support these student compilers also support the more conventional compilers which store the object code.

Examples and Discussion:

It was shown by Jacopini and Bohm¹ in 1966 that all algorithmic procedures (e.g., computer programs) can be represented in terms of two elements - actions and decisions - which form three basic structures: A sequence of actions, a condition construct (IF condition THEN action1 ELSE action2), and a loop construct (WHILE condition DO action). It was later demonstrated² that limitation of algorithmic development to these three constructs produced a more reliable software. Later, Ledgard³ extended the class of reliable language structures to include single branching conditional constructs (IF condition THEN action), n-way branching conditional constructs (the CASE statement), and a variant loop construct (DO action UNTIL condition) which always performs the specified action at least once.

Such findings would be completely academic to the average person concerned with data processing, except that it has been amply proven that limitation to these structures produces a program that is lower in development and maintenance costs. For this reason, the United States Government has sponsored the fifteen volume Structured Programming Series, of which this report is an extension. Section 2.2, Volume I of the SPS specifies the main features of structured programming and top-down programming while Section 3 of the same volume defines the TDSP standards and techniques for implementation.

¹ Bohm, C. and Jacopini, G., "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules", Communications of ACM, Vol 9, No. 4, May 1966.

² Dahl, W.M., Dykstra, E.W., and Hoare, C.A.H., "Structured Programming", Academic Press, New York, 1972.

³ Legard, H.F., "A Genealogy of Control Structures", Communications of the ACM, Vol. 18, No. 11, November 1975.

Example 3.3.1.a.1: Compilation and object module storage in a Type I system.

Figure 3.3.1.a-1 below shows the compilation of a FORTRAN program and the storage of the resulting object module. "R F" requests the FORTRAN compiler. Next, the output and input files are specified. Since extensions aren't given in either case, the default assumption is that they are OBJ and .FOR respectively. PIP is then called to verify file existence.

```
.R F
*B55SAM=B55SAM
*!C

.R PIP
*B55SAM.**/L
21-SEP-76
B55SAM.FOR      1 16-SEP-76
B55SAM.OBJ      3 21-SEP-76
2 FILES, 4 BLOCKS
3127 FREE BLOCKS
*
```

Figure 3.3.1.a-1. Compiling of source code and storing of the resulting object module by a Type I system.

Example 3.3.1.a.2: Compilation and object module storage in a Type II system.

This time sharing system offers several compilers which may be invoked simply by typing their name, the source file name, and any valid compiler options desired. Such options may include automatic execution-time validity checking of subscripts, suppression of object code creation (i.e., syntax and allocation checking only), and specification of level of machine code optimization. Figure 3.3.1.a-2 shows the compilation of the FORTRAN program B55 SAMPL and subsequent creation of its object module (assigned the file type of "TEXT").


```

15.37.02 >FORTRAN B55SAMPL

15.38.14 >LISTF B55SAMPL *
FILENAME FILETYPE MODE ITEMS
B55SAMPL FORTRAN   P   313
B55SAMPL TEXT      P    27

15.29.43 >

```

Figure 3.3.1.a-2. Compiling a source file and automatically storing the object module on a Type II system.

Example 3.3.1.a.3: Compilation and object module storage in a Type III system.

Compilation and object module storage are typically more complex operations in this system. As can be seen in Figure 3.3.1.a-3 below, first the source file must be attached and assigned a local name, then the FORTRAN compiler is run with the FTN command and specification of the local file name of the input. The object module is placed in a local file called LG0 and must be specifically saved in order to prevent its loss.

COMMAND- ATTACH,X,B55SAMPLEFORTRAN,ID=GAERTNER

PF CYCLE NO.= 001
COMMAND- FTN,I=X

0.124 CP SECONDS COMPILATION TIME
COMMAND- CATALOG,LGO,B55SAMPLETEXT,ID=GAERTNER

INITIAL CATALOG
RP = 090 DAYS
CT ID= GAERTNER PFN=B55SAMPLETEXT
CT CY= 001 0000256 WORDS.:
COMMAND-

Figure 3.3.1.a-3. Compiling a source file and storing the resultant object module on a Type III system.

3.3.1.b Load Module Generation

Classification: Basic

Requirement:

Load Module Generation - The basic implementation of a PSL must provide the interface required to execute those programs necessary for the conversion of object modules into executable code. The PSL must also provide a means of storing the resulting machine code in the library or of loading the program(s) into storage for subsequent execution.

Purpose of Paragraph:

To specify the need to generate and store program load modules.

Existing Tools Which Satisfy This Requirement:

All known systems are capable of generating executable machine code which of necessity is stored at least until the termination of job execution. The facility for performing this task is usually known either as a Linker or a Loader.

Examples:

The capability to generate load modules and store them, either permanently or temporarily just prior to execution, is basic to all computer installations. Following are examples illustrating how this capability is implemented in the three representative systems. Linkage of a FORTRAN program has been chosen as the subject of the examples, but methodologies are similar for other languages, both assembler and high level.

Example 3.3.1.b.1: Generation and storage of load modules in a Type I system.

In the PDP-11 system used to represent Type I, the LINK system command is used to coordinate previously generated object modules with the necessary elements from the system and the language library. The linker is capable of producing load modules on three different formats: (1) Pure memory image for use in a single-job system or in background of a Foreground/Background system, (2) an alternate memory image for use with the absolute loader, and (3) a relocatable image for foreground execution in a Foreground/Background system. The load module format as well as other linker options are specified by "switches" (single letter commands) which may be appended to the command line which defines the input and output files. Figure 3.3.1.b-1 shows the use of system defaults in linkage of the FORTRAN program B55SAM. If it had been fully specified the command would have read "DK:B55SAM.SAV=DK:B55SAM.OBJ/F", where "/F" designates that the FORTRAN library is to be used. A load map could also have been requested.

After the program has been linked, PIP is used to verify the existence of the memory image file (extension name .SAV), then the program itself is executed ("R B55SAM").

```

•R LINK
*B55SAM=B55SAM/F
*IC

•R PIP
*B55SAM.SAV/L
17-SEP-76
B55SAM.SAV 9 17-SEP-76
*IC

•R B55SAM

```

INPUT DATA

1.010	3.342	10.959	35.678	15.439
71.531	90.235	97.627	73.649	63.250
16.659	30.709	34.320	29.542	68.369
44.335	50.686	5.103	74.447	0.753

Figure 3.3.1.b-1. Creating a load module that is saved on disk and executing it on a Type I system.

Example 3.3.1.b.2: Generation and storage of load modules in a Type II system.

Loading and linking of all necessary object modules is done via the LOAD command in CSS and CMS. In both systems, a load map is automatically generated unless specifically suppressed. CSS requires the user to specify all subroutines either in the command line or in the USE command which extends the LOAD command. In CMS, the system automatically searches all user disks to resolve external references. Both systems require a prior issuance of the GLOBAL command if libraries other than the system library are to be searched.

If a permanent copy of the core image is desired, the user may issue the GENMOD command (valid in both CMS and CSS) as shown in Figure 3.3.1.b-2. Note that actual program execution does not begin until the START command is given. Both systems have a load option for immediate execution; and, both require the LOADMOD command to bring the core image file back into memory after a GENMOD command has been given.

```

15.13.33 >LOAD B55SAMPL
15.14.01 >GENMOD B55SAMPL
15.14.41 >LISTF B55SAMPL MODULE
B55SAMPL MODULE      P      2
15.16.05 >LOADMOD B55SAMPL
15.16.43 >START

```

INPUT DATA

0.427	1.685	6.262	22.412	78.113
66.969	98.796	90.053	51.151	96.434
18.246	41.567	85.188	37.022	55.437
99.429	97.641	90.982	67.123	83.901

Figure 3.3.1.b-2. Creating a load module that is saved on disk and subsequently recalled for execution on a Type II system.

Example 3.3.1.b.3: Generation and storage of load modules in a Type III system.

Prior to actual loading of a program or set of programs, the CDC SCOPE system requires that all libraries that are to be searched for external references first be specified. This is done via the LIBRARY and LDSET(LIB=) commands. Only the system library, NUCLEUS, is automatically included in searching.

Once libraries have been named, a specific request must be made to save the memory image load module if that is desired. This is done by requesting a permanent file device for the given logical file name ("REQUEST, SAMPLE, *PF"). At this point the user may attach all the object modules which are necessary and issue the XEQ command. The system response is to prompt for input of execution options such as the names of other object files or libraries. The prompt is terminated by giving a NOGO, EXECUTE, or file name response. The dialogue of Figure 3.3.1.b-3 first shows the use of the NOGO option to force loading completion without execution. This allows the core image to be saved via the CATALOG command. While the core image file does not have to be retrieved

again (as in the Type II system) it does have to be rewound to reposition it to its starting point. The XEQ command now causes program execution.

Note that there is a great deal of redundancy required in loading and execution of programs containing external references. All libraries involved must be specified both in an initial LIBRARY or LDSET command, then again as the arguments for the LIBLOAD option of the XEQ command. The files to be used, whether in a library or not, must be specifically attached to the user's facility (ATTACH command). If a file is not part of a library but is to be referenced, it is specified again in the XEQ LOAD option. The program loading the execution is a far more tedious process in this system than in the others.

COMMAND- REQUEST, SAMPLE, *PF

COMMAND- ATTACH, X, B55SAMPLETEXT, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- XEQ

OPTION=LOAD=X

OPTION=NOGO

COMMAND- CATALOG, SAMPLE, SAMPLE, ID=GAERTNER

INITIAL CATALOG

RP = 090 DAYS

CT ID= GAERTNER PFN= SAMPLE

CT CY= 001 00000128 WORDS.:

COMMAND- REWIND, SAMPLE

COMMAND- XEQ, SAMPLE

INPUT DATA

0.427	1.685	6.262	22.412	78.113
66.969	98.796	90.053	51.151	96.434
18.246	41.567	85.188	37.022	55.437
99.429	97.641	90.982	67.123	83.901

COMMAND-

Figure 3.3.1.b-3. Creating a load module which is saved on disk and later executed on a Type III system.

3.3.2 Full Programming Language Support Requirements

Classification: Full

Requirement:

Full Programming Language Support requirements include all of the Basic Programming Language Support requirements and the following additional requirements.

Purpose of Paragraph:

To introduce additional requirements for the Full implementation of PSL Programming Language Support facilities.

3.3.2.a Full Top Down Structured Programming Support

Classification: Full

Requirement:

Full Top Down Structured Programming Support - The PSL must include facilities which provide a capability to perform, at user option, automatic exception checking of certain top down structured programming standards which are more efficiently checked during library update than during compilation. Such exception checking must include but is not limited to:

1. Flagging (i.e., identifying on a listing) of all explicit branches such as GO TO statements.
2. Flagging program language source code files or programs blocks that exceed a maximum size to be defined by the user.
3. Flagging any lines of source code that contain more than one source statement (a line of code is defined as one source file record). This requirement is based on Coding Standard 4 presented in Section 3.2 of Volume I of the Structured Programming Series which states that "only one statement per line of code is permitted". The reasons for this restriction are increased program readability and suitability for pre-compiler input.

Examples:

There are currently no facilities for exception checking in any of the three systems studied, nor do commercially available precompilers offer such checking. It would be necessary for the contractor to either have access to one of the pre-compilers previously mentioned or to acquire or create additional software. Program specifications for such software may be found in the following section.

Requirements for Necessary New Support Software:

The following are initial program specifications for a set of routines to implement automatic exception checking for a given source code file in a given language. These specifications assume that a precompiler is available which allows use of the key control structures. If that is not true, then additional data structures and actions would be required to separate structured from non-structured uses of explicit branches. (For a summary of structured use of explicit branching see SPS Vol. II, Section 4.) These specifications are not intended to be final or complete.

Input Parameters:

- 1) Source file designation,
- 2) Source language.

Key Variables and Data Structures:

In order to create a flexible program that may be used for several source languages, and that may be readily expanded to accommodate more supported languages, the key data structure is a set of tables. Each table entry contains a table entry number (TAB_ENT_NO) which is used to coordinate the tables; all table entries with the same number are concerned with the same source language.

The tables as initially designed have primarily variable length entries. If a contractor considers time limitations of greater import than space, then these tables may be made fixed length and the specifications altered accordingly.

- 1) Language Table - A table which contains the names of all languages which can be handled by this program. Each fixed length entry would consist of:
 - a) byte 1 table entry number (TAB_ENT_NO) an integer indicating if first language in table, second language, etc.
 - b) bytes 2-12 Character string data which is the source language name, e.g., COBOL.
- 2) Branching Keyword Table - A table which specifies the keywords used for explicit branching in all supported languages. Since some languages have more than one form of keyword for this function, the entries are variable length, each having the following structure:

byte 1 TAB_ENT_NO

byte 2 length of table entry in bytes

byte 3 number of keywords in entry (NK)

byte 4 byte NK+3 length of each keyword (LK)

byte NK+4 - byte NK+LK(1)+4 first keyword.

There will be NK consecutive keywords.

For example, if language #2 has two forms of an explicit branch, "GO TO" and "GOTO", then its table entry would be as follows:

2	14	2	5	4	G	O	T	O	G	O	T	O
---	----	---	---	---	---	---	---	---	---	---	---	---

where each marked segment () designates one byte.

3. Maximum Unit Size Table - For each language used at an installation, the maximum size for each defined unit type. This table would be entirely user defined. Since there are several ways of segmenting source code for production analysis, this table allows for several user-defined units to be checked concurrently. Units are defined in terms of character strings used as unit delimiters. Unit types are given numbers such that unit type 0 is the entire file (no character string check needed), while the remaining types are represented by higher integers which are arbitrarily assigned. Languages which are not used at the given installation should contain all 0 entries. The entry structure might appear as:

byte 1 TAB_ENT_NO

byte 2 length of table entry in bytes

byte 3 number of types of units defined for this entry (NTU)

byte 4 - NTU+3 integers representing unit types defined for given installation

byte NTU+4 - 3*NTU+3 maximum unit size in records for each unit as subsequently defined. Expression for size is limited to two bytes, thus the maximum expressible unit size is $2^{16}-1$ or 65535 records

byte 3*NTU+4 length in bytes of beginning delimiter for first unit type

byte 3*NTU+5 length in bytes of terminal delimiter for first unit type. There will be NTU such byte pairs.

byte 5*NTU+4 beginning and terminal delimiters for each unit type. There will be NTU such delimiter pairs.

For example, if a program coded in language #4 were to be checked for size on three unit types - file, begin block, and procedure - which have type numbers, delimiters and maximum sizes as in Table I, then the MUS Table entry would appear as follows:

TABLE I

max. size	unit type	unit code	delimiters
1000	file	0	
150	begin block	2	BEGIN, ENDBEGIN
300	procedure	3	PROC, ENDPROC

MUS Table Entry

```

4 , 42 , 3 , 0 , 2 , 3 , 1000 , 150 , 300 , 0 , 0 , 5 , 8
4 , 7 , B , E , G , I , N , E , N , D , B , E , G , I , N , P
R , 0 , C , E , N , D , P , R , O , C

```

Note: Each Segment represents 1 byte

- 4) Source Line Incompatibility Table - For all supported languages, those character strings which may not exist within the same record of structured source code. These may be viewed as incompatibility pairs. For examples, in structured FORTRAN, "IF" and "GOTO" should occur on separate lines; in PL/1, the occurrence of two statement delimiters (a semicolon) on the same line would cause flagging. Each variable length entry would appear as follows:

```

byte 1  TAB_ENT_NO
byte 2  length of table entry in bytes
byte 3  number of incompatible pairs in this language
        (NI)
byte 4  length in bytes of first element of first pair

```

byte 5 length in bytes of second element of first pair. There will be NI byte pairs expressing element lengths.

byte $2*NI+4$ end of entry of incompatibility string pairs.

For example if language #1 cannot contain the words "IF" and "ELSE" on the same line, nor have two semicolons, then the SLI Table entry would be:

1	15	2	2	4	1	1	1
F	E	L	S	E	;	;	

- 5) FLAG_CNT - A 3 element integer array which contains counts of occurrences of each of 3 error types.
- 6) OFFSET - An integer which is the number of bytes into a table that an element of interest begins
- 7) CURR_BYTE - A character which is the contents of the byte being currently pointed to by OFFSET in the TABLE-LOCN subroutine.
- 8) ERROR-FLAG - A logical variable which indicates that the entry for a given language was not found in the table.
- 9) NO_BRCH_KEYS - An integer giving number of branching keywords.
- 10) BRANCH_KEY - A character string array which can accommodate NO_BRCH_KEYS elements, each of which is a branching keyword.
- 11) NO_INCMPT_STRNGS - An integer giving number of incompatible source line string pairs.
- 12) INCMPT_STRNG - A two-dimensional character string array which can accommodate NO_INCMPT_STRNGS x 2 elements, each of which is a member of the incompatible string pair. The first subscript designates the number of the pair, the second subscript indicates if the first or second member of the pair.
- 13) NO_UNITS - An integer giving number of units defined in the Maximum Unit Size Table.

- 14) UNITS - A data aggregate array which can accommodate NO UNIT elements, each of which stores data on a specific type of unit to be checked for size. Each aggregate element has the following structure:
- 1 UNITS
 - 2 TYPE integer
 - 2 MAX_SIZE integer
 - 2 BEGINNING_DELIMITER character string
 - 2 TERMINAL_DELIMITER character string
 - 2 ENTERED logical
 - 2 BEGINNING_LINE_NO. integer
 - 2 LINE_COUNT integer.
- 15) FOUND - A logical variable indicating if a string is contained within another string.
- 16) WD_START - Integer giving starting byte of a character string within a table.
- 17) INDEX - An integer used to indicate the number of times through a loop.
- 18) N - An integer variable with same function as INDEX
- 19) LENGTH_PTR - A 2-element integer array containing pointers to LENGTH values
- 20) LENGTH - A 2-element integer array containing length values of character strings.
- 21) TYPE_PT - An integer variable which points to location of TYPE.
- 22) SIZE_PT - An integer variable which points to location of MAX_SIZE.

Program Performance:

Main Routine:

- 1) Read in language table.
- 2) Search language table for source language.
- 3a) If source language not in table, issue an error message and exit from program.
- 3b) Else continue. Set TAB_ENT_NO to index of language within Language Table and read in Branching Keyword Tables (BK_TABLE), Maximum Unit Size Table (MUS_TAB) and Line Source Incompatibility Table (SLI_TABLE). The storage for these should be byte addressable. Initialize all necessary variables. Storage for Language Table may be released now.
- 4) Extract needed information from tables
 - a) Call TABLE_LOCN subroutine
Input: BK_TABLE and TAB_ENT_NO
Returned: ERROR FLAG and OFFSET
 - b) If ERROR FLAG clear then
 - i) NO_BRCH_KEYS ← contents of BK_TABLE
byte OFFSET + 3
WD_START ← NO_BRCH_KEYS + OFFSET + 4
LENGTH_PTR(1) ← 3
INDEX ← 3
Repeat following NO BRCH KEYS times:
Increment INDEX by 1
Increment LENGTH_PTR(1) by 1
LENGTH(1) ← contents of byte LENGTH_PTR(1)
BRANCH KEY (INDEX) ← character string starting
at WD_START of BK_TABLE for length of LENGTH.
WD_START ← WD_START + LENGTH.

- ii) Else issue error message and NO BRCH KEYS 0.

- c) Call TABLE_LOCN subroutine.
 Input: SLI_TABLE and TAB_ENT_NO
 Returned: ERROR FLAG and OFFSET

- d) If ERROR FLAG clear, then
 - (i) NO_INCMPT_STRINGS ← contents of SLI_TABLE byte
 OFFSET + 3
 WD_START ← NO_INCMPT_STRINGS * 2 + OFFSET + 4
 LENGTH_PTR(1) ← OFFSET + 2
 LENGTH_PTR(2) ← OFFSET + 3
 INDEX ← 0
 Repeat following NO_INCMPT_STRINGS times:
 Increment INDEX by 1
 Increment both elements in LENGTH_PTR by 2
 LENGTH(1) ← contents of SLI_TABLE byte LENGTH_PTR(1)
 LENGTH(2) ← contents of SLI_TABLE byte
 LENGTH_PTR(2)
 INCMPT_STRNG(INDEX,1) ← character string
 starting at WD_START of SLI_TABLE for
 length of LENGTH(1)
 WD_START ← WD_START + LENGTH(1)
 INCMPT_STRNG(INDEX,2) ← character string
 starting at WD_START of SLI_TABLE for
 length of LENGTH(2)
 WD_START ← WD_START + LENGTH(2)

 - (ii) Else issue error message and
 NO INCMPT_STRNGS ← 0

e) Call TABLE_LOCN subroutine
 Input: MUS_TABLE and TAB_ENT_NO.
 Returned: ERROR_FLAG and OFFSET

f) If ERROR_FLAG clear then

(i) NO_UNITS ← contents of MUS_TABLE byte OFFSET + 3
 TYPE_PT ← OFFSET + 3
 SIZE_PT ← NO_UNITS + OFFSET + 2
 LENGTH_PTR(1) ← 3*NO_UNITS + OFFSET + 2
 LENGTH_PTR(2) ← LENGTH_PTR(1) + 1
 WD_START ← 5*NO_UNITS + 4
 INDEX ← 0
 Repeat following NO_UNITS times:
 Increment TYPE_PT by 1,
 SIZE_PT by 2,
 LENGTH_PTR Array by 2
 INDEX by 1
 LENGTH(1) ← contents of LENGTH_PTR(1)'th
 byte of MUS_TABLE
 LENGTH(2) ← contents of MUS_TABLE byte
 LENGTH_PTR(2)
 TYPE(INDEX) ← contents of MUS_TABLE byte
 TYPE_PT
 MAX_SIZE(INDEX) ← contents of MUS_TABLE byte
 SIZE_PT
 ENTERED(INDEX) ← 0
 BEGINNING_LINE_NO(INDEX) ← 0
 LINE_COUNT(INDEX) ← 0
 If TYPE(INDEX) > 0 then perform following:
 BEGINNING_DELIMITER(INDEX) ← character string
 starting at WD_START of MUS_TABLE for length
 of LENGTH(1).

TERMINAL_DELIMITER(INDEX) ← character string
starting at WD_START of MUS_TABLE for length
of LENGTH(2) WD_START ← WD_START + LENGTH(2)

(ii) Else issue error message and NO_UNITS ← 0

g) Storage space for tables may now be freed. Read source
code file into buffer.

INDEX ← 0

h) Repeat following until end of source file:

Increment INDEX by 1

Read INDEX'th line (i.e., record) of source file.

If end of file, then

(i) print out number of lines flagged for each error
type, i.e., contents of FLAG_CNT

(ii) Else perform following:

(1) N ← 0

Repeat following NO_BRCH_KEYS times:

Increment N by 1

Call STRING-COMPARE subroutine

Input: BRANCH_KEY(N) and current source line

Returned: FOUND

If FOUND is true then

Increment FLAG_CNT(1) by 1

Print out statement of error type, line
number and line image.

(2) $N \leftarrow 0$

Repeat following NO_INCMPT_STRNGS times:

Increment N by 1

Call STRING_COMPARE subroutine:

Input: INCMPT_STRNG(N,1) and current source
line

Returned: FOUND

If FOUND is true then perform following:

Call STRING_COMPARE subroutine

Input: INCMPT_STRNG(N,2) and current
source line

Return: FOUND

If FOUND is true then

Increment FLAG_CNT(2) by 1 and

Print out statement of error type,
line number and line image.

(3) NOTE: Algorithm assumes that nested units of same
type are not allowed.

Repeat following NO-UNITS times

Increment N by 1

Call STRING_COMPARE subroutine

Input: BEGINNING_DELIMITER(N) and current
source line

Returned: FOUND

If FOUND is true then

(a) ENTERED(N) ← true

BEGINNING_LINE_NO(N) ← INDEX

LINE_COUNT(N) ← 1

(b) Else

Call STRING_COMPARE subroutine

Input: TERMINAL_DELIMITER(N)

current source line

Returned: FOUND

If FOUND is true then

ENTERED ← false

Else increment LINE_COUNT(N) by 1

If LINE_COUNT(N) > MAX_SIZE(N)

and ENTERED is true then print error

message, BEGINNING_LINE_NO, INDEX, LINE_COUNT,

unit TYPE, etc.

End of h) loop

Subroutines:

1) TABLE_LOCN subroutine:

Input: A table and TAB_ENT_NO

Returned: ERROR_FLAG and OFFSET

Performance:

ERROR_FLAG ← 0

OFFSET ← 1

Repeat following until end of table reached:

```

CURR_BYTE ← contents of OFFSET'th byte of table

If CURR_BYTE=TAB_ENT_NO then
    OFFSET ← OFFSET - 1 to adjust and return.
Else
    OFFSET ← OFFSET + contents of (OFFSET + 1)'th
    byte of table, i.e., entry length added to OFFSET
    to reposition table pointer
ERROR_FLAG ← 1
Return

```

2) STRING_COMPARE subroutine

Input: A character string being searched for and a
the line being searched

Returned: Logical variable FOUND.

Performance:

FOUND ← 0

Search for occurrence of parameter 1 within
parameter 2.

NOTE: Since languages vary greatly in string search
facility, mode of search is not specified.

If search is successful the FOUND ← true
Return.

Input Summary:

Variable input which is specified with each program
use: Source file designation and source language.

Constant input:

Language Table

Branching Keyword Table

Maximum Unit Size Table
Source Line Incompatibility Table.

Output Summary:

If source language specified is not supported a statement of such is printed and program terminates.

If source language specified has not been fully implemented in any of the Tables then a statement of such is printed and that error type will not be handled.

If an explicit branch is found, printout states that error is explicit branch, gives line number and line image.

If two incompatible syntax elements are found in the same line, then printout states that there is non-structured syntax in line, gives line number and line image.

If a defined unit size is exceeded, printout states that error, gives unit type, maximum size, beginning line number of unit, and number of line which triggered size exception.

A final summary of counts of three error types is printed at end of program.

3.3.3 Summary of Facilities for Programming Language Support

Basic programming language support facilities are present in virtually all computer installations. The capabilities to compile or assemble a source program, store the compiled version, and link and execute the program are too vital not to be present. The major difference between systems then, is the quality of these basic software components. While it is impossible for the average contractor to have the most recent compiler in the most sophisticated language, it is thoroughly feasible for him to set time aside for system maintenance and installation of up-to-date compiler versions sent by the software vendor. Such update activities are frequently ignored in production-oriented computer facilities, but hopefully the contractor will find that improved software is cost beneficial in the long run.

Precompilers, though mentioned throughout the original SPS report, are not widely available on a commercial (as opposed to experimental) basis. The ones that are best known, such as RATFOR, still do not follow all the standards specified for precompilers in the SPS report, as none perform exception checking. It is suggested that the Government encourage the production of commercial compilers if such tools are to be considered key to a structured programming support library.

Of the three computer systems generally discussed, Type I and Type II both offer relatively easy means of compiling and storing a program. The Type III system proves to be unwieldy during program loading and execution, though it still fulfills the basic requirements for programming language support. None of the systems has standard software that performs exception checking, nor is such software commonly available. It is suggested that the Government provide such software to those contractors who have need of Full PSL implementations.

3.4 Library System Maintenance

Requirement:

Library System Maintenance - This functional area involves the generation and maintenance of the library system and related data storage and indices. The basic requirements include a facility to install, maintain and terminate the PSL. The full requirements expand this facility to provide a system generation capability.

Purpose of Paragraph:

To introduce the Library System Maintenance functional area and to delimit the scope of the Basic and Full requirements within this area.

3.4.1 Basic Library System Maintenance

3.4.1.a Library Installation Support

Classification: Basic

Requirement:

Library installation support - A procedure and the related system support facilities must be provided to install the PSL at a user location.

Purpose of Paragraph:

To specify the need for a means of installing a PSL and to indicate the types of software required.

Discussion of PSL Contents and Organization

Any discussion of the installation of a PSL must first be prefaced by an enumeration of the PSL contents and of other software assumed to exist prior to PSL installation. For such, the reader is directed to SPS Volume V, Section 2 and Volume VI, Sections 1 through 4. To review that material briefly, pre-existing software is assumed to include:

Operating System,
Direct access storage device allocation program,
Assembler,
Compiler(s),
Link Program,
Loader,
Storage manipulation facilities, e.g., PIP,
Editor.

The PSL itself consists of three segments: PSL support software, catalogues and directories, and files.

PSL Support Software:

Programs which are frequently part of a system that are needed for PSL support (e.g. editors) have been included in the preceding list of pre-existing software. Other software support is described in detail within those sections of this report which pertain to the functional areas which require said support. It is assumed that the Government will supply its contractors with the support programs written in languages which the contractors' facilities are capable of handling.

Catalogues and Directories:

Aside from the standard device directories, the PSL must include (i.e., as part of Basic requirements) the PSL Catalogue, Project Catalogue, and Library File Index. If Full PSL implementation is being considered, then a Name Table File Index and Management File Index must also be included. Further descriptions of these catalogues may be found in SPS Volume VI section 4.2.2. However, the reader should note that terms used in that discussion differ from those used elsewhere. The nature of those differences are discussed in the following section.

Files:

In the majority of installations a file is considered the basic unit. It is that unit which is listed in device directories. Typical examples of a file are a single source code program, object code of a single program, or text documenting a single aspect of a project. A file is logically

cohesive unit. In SPS Volume VI, the term used for this is "unit".

SPS Volume VI, Section 4.2 on Library Organization defines file organization as consisting of project name, library name, file name and unit name. Analysis of terms reveals that "library name" refers to a project subdivision name that indicates a programming system version number or purpose, whereas "file name" refers to the file type as it is normally considered, e.g., source, object, text, or job control language. This is often referred to as the library designation. Since it is considered advisable by most to maintain standard usage of terms to prevent confusion, such standard usage will be used within this report. Thus the PSL organization may be viewed as consisting of project name, project subdivision name, file type name, and file identifiers. A complete file designation should refer to all levels of organization. Such file organization can be implemented in the three representative systems as shown in the discussion associated with Section 3.2.2.d. A brief review follows.

Type I Systems - File designation consists of a maximum six letter name and a three letter extension (e.g., PROGRAM.SRC). Since programmer designation needs to be included, the first letter of the file name could be programmer code, second letter the project code, third letter the subdivision code and fourth through sixth letters the file identifier. The extension is used by the system to show file type.

Type II System - File designation consists of a maximum eight letter file identifier, a maximum eight letter filetype, and a one letter storage device designation. Storage devices could be individually designated to projects and since filetype i.d. is used by the system to indicate filetype (e.g., FORTRAN, COBOL, TEXT), that leaves the file identifier for further specification. The first letter could be used to identify programmer, the second and third characters to indicate project subdivision, and the fourth through eighth letters the file name.

Type III System - While the CDC SCOPE system does allow user creation of libraries, those libraries are limited to containing only object modules, i.e., assembled machine code. Therefore, in order to implement the necessary hierarchical file structure, the best method is to encode the 40 character file name. The user ID may be used to indicate the project name, while programmer name, project subdivision, file type, and file identifier may be indicated in the 40 characters allotted.

Example of PSL Installation:

The installation operation involves allocating space for the PSL Catalogue and subsidiary catalogues, initializing those catalogues by writing the first and last entries, and compiling the PSL software. This latter item should be supplied by the Government to the contractor as part of the PSL package. These PSL support programs would form one project division, which could be subdivided according to support function. They would need to be stored in source, object module, and load module form.

The exact methods for installing the PSL will vary with the supporting operating system and hardware configuration. The first step would be to compile and execute the Library System Maintenance (LSM) program that would be supplied (see SPS Volume VI, Section 5.1). After that, the INSTALL function of the LSM could be used to start catalogue generation. It must be remembered that the PSL Catalogue is not a directory like the device directory used to list files, but rather is "a data area used to record the location of the Name File Table, if used, and for each project, the location of its Project Catalogue". Thus the catalogue is, as far as the operating system is concerned, just another file. The variance in file manipulation procedures between computer facilities will require the LSM program to be highly interactive, so that the system-specific details may be given at installation time.

The beginning phases of PSL implementation by contractors are bound to be chaotic, but careful planning, training of personnel, and free interchange of information between Government and contractor should ease the situation. Thorough notes should be taken on all procedures used, and feedback given to the writers of the PSL software support package. As a wider variety of installations use the package, definite methodologies can be set up for each type of supporting facility.

3.4.1.b Library Maintenance Support

Classification: Basic

Requirement:

Library Maintenance Support - The following support of data storage maintenance must be provided.

1. Allocation of new library data storage and directory space.
2. Reallocation of existing library data storage and directory space.
3. Compression of data in existing files to eliminate deleted or superseded data and recover unused space.

Purpose of Paragraph:

To specify maintenance operations which are necessary for the physical storage of data and directories.

Existing Tools which Satisfy this Requirement:

In many systems, allocation of storage space for data files (and for library catalogues as well) is an automatic system function. If the file is revised, space is reallocated, also automatically, or additional space assigned to the latest version.

Compression of data is not typically automatic, though it is in several Type II facilities. Most often data space recovery is achieved by copying all active files onto an intermediate storage device then copying back to the original storage device. Sometimes utility programs such as PIP handle this function, and sometimes it is necessary for the entire operation to be user directed.

Examples:

As was noted at the end of the Examples for SPS 3.4.1.a, the PSL Catalogue and subsidiary directories appear as data files to the operating system. Therefore, the following discussion refers to both data files and directories.

Example 3.4.1.b.1: Allocation, reallocation and compression of data storage space in a Type I system.

These functions are performed by the file manipulation program resident on the given system. For Hewlett-Packard's RTE IV, FMP (File Management Package) is used. For DEC's RT-11 PIP is used. While the exact syntax varies slightly, the functions each program can perform are similar. The DEC system is used in the following example.

Initial allocation of storage space depends upon the file type and mode of generation. A file containing source code, written text, or Program Design Language statements is normally created using the editor's Edit Write command. Normally, half the largest available contiguous space on the disk is allocated. However, the user may specify the space allocation in blocks following the extension designation of the output file, as is shown in the Figure 3.4.1.b-1. If at some later time the original allocation proves insufficient, space may be reallocated by use of the "/T" switch, provided that sufficient additional space exists following the file. If it doesn't, the space may be created by using the "/X" switch to move other files.

If the file is an object module it is created by the compiler or assembler and stored by the system, being allocated only the space it needs, the same applies to load modules.

Compression of files and directories is done by user specification of the device and the "/S" switch.

In the example below (Figure 3.4.1.b-1) the user enters PIP and obtains a directory listing. He then edits a FORTRAN program, specifying the output file block size so that the first unused space of sufficient size will be used for storage. Following editing, a directory listing is again obtained to show that the original BAFRAN.FOR has been renamed to indicate that it is a backup copy, and that the 2 block unused space rather than the 6 block space was used for storage of the new version. Since the object and load modules are now out of date, they are deleted. Next more room is allocated to the PDL file for BM1CCD via the "/T" switch. Since there is sufficient unused space following the file, it is not necessary to reallocate space for other files. However, such reallocation is necessary in order to extend the size of BM1CCD.TXT. By copying BM1CCD.FOR (the file after .TXT) back onto DK using the "/X" switch, BM1CCD.FOR is moved into the first clear space, thus leaving room for the .TXT file to be extended.

Any catalogues associated with these files would have to be updated by either the user using the editor or by additional software provided for that purpose.

```
.R PIP
*/E/C
23-MAR-73
BAFRAN.FOR      2 23-MAR-73
BAFRAN.OBJ      3 23-MAR-73
BAFRAN.SAV      3 23-MAR-73
< UNUSED >      2
BMICCD.PLD      1 23-MAR-73
< UNUSED >      6
BMICCD.TXT      2 23-MAR-73
BMICCD.FOR      2 23-MAR-73
< UNUSED > 3357
6 FILES, 11 BLOCKS
3365 FREE BLOCKS
```

```
*IC
.R EDIT
*EBBAFRAN.FOR(2)SR$$
*
```

*E<\$\$

```
.R PIP
*/E/C
23-MAR-73
BAFRAN.BAK      2 23-MAR-73
BAFRAN.OBJ      3 23-MAR-73
BAFRAN.SAV      3 23-MAR-73
BAFRAN.FOR      2 23-MAR-73
BMICCD.PLD      1 23-MAR-73
< UNUSED >      6
BMICCD.TXT      2 23-MAR-73
BMICCD.FOR      2 23-MAR-73
< UNUSED > 3357
7 FILES, 13 BLOCKS
3363 FREE BLOCKS
*BAFRAN.OBJ,BAFRAN.SAV/D
*BMICCD.PLD(2)=/T
*/E/C
```

```
23-MAR-73
BAFRAN.BAK      2 23-MAR-73
< UNUSED >      6
BAFRAN.FOR      2 23-MAR-73
BMICCD.PLD      2 23-MAR-73
< UNUSED >      5
BMICCD.TXT      2 23-MAR-73
BMICCD.FOR      2 23-MAR-73
< UNUSED > 3357
5 FILES, 10 BLOCKS
3363 FREE BLOCKS
*BMICCD.FOR=BMICCD.FOR/
*BMICCD.TXT(3)=/T
```

Figure 3.4.1.b-1. Allocation, Reallocation and Compression of data storage space in a Type I system.

```

*/E/C
23-MAR-73
BAFRAN.BAK      2 23-MAR-73
BMICCD.FOR      2 23-MAR-73
< UNUSED >      4
BAFRAN.FOR      2 23-MAR-73
BMICCD.PLD      2 23-MAR-73
< UNUSED >      5
BMICCD.TXT      3 23-MAR-73
< UNUSED > 3353
5 FILES, 11 BLOCKS
3367 FREE BLOCKS
*SF: /S
REBOOT?
*/E/C
23-MAR-73
BAFRAN.BAK      2 23-MAR-73
BMICCD.FOR      2 23-MAR-73
BAFRAN.FOR      2 23-MAR-73
BMICCD.PLD      2 23-MAR-73
BMICCD.TXT      3 23-MAR-73
< UNUSED > 3367
5 FILES, 11 BLOCKS
3367 FREE BLOCKS
*IC

.REBOOT
*IC

```

Figure 3.4.1.b-1 (continued). Allocation, Reallocation and Compression of data storage space in a Type I system.

Example 3.4.1.b.2: Allocation, reallocation and compression of data storage space in a Type II System.

Both of the commercial time-sharing systems investigated are virtual machine based. As such, allocation of storage is dynamic and constantly readjusted to the user's needs. The operating system itself tends to such matters within the scope of the user's machine configuration, always maintaining optimal spatial usage. If there is insufficient total disk space available, additional space may be temporarily gained via the CSS command ATTACH or the CMS commands DEFINE and ACCESS. Permanently increased disk space may be obtained upon petition to the manager of the time-sharing facilities.

The only control an individual user has over his file storage space is the result of his control over the file's formats. Both sample systems allow files to be either packed or normal, and to have record length user-defined upon creation. CMS further permits record length to be latter altered via the copyfile command.

Example 3.4.1.b.3: Allocation, reallocation and compression of data storage space in a Type III System.

While the Type III system accessed is not virtual, its operating system also maintains optimal use of storage space; allocation and reallocation of space is automatic, as is compression of files into the minimum amount of space. In CDC SCOPE, this is accomplished by segmenting a file over perhaps more than one device. All files are "recorded in a logical sequence of record blocks which may be arbitrarily scattered about on the disk surface. SCOPE maintains a central memory table for each file ... in which the sequence of allocated record blocks is defined". The allocation can be increased by the user via the EXTEND and ALTER functions.

Unlike the Type II systems, SCOPE gives the user no control over the format of his files, other than altering tab settings. Record length is a function of file type, and is defined separately for each installation.

¹ Scope Reference Manual, Control Data Corporation, 1974, page 3-14.

3.4.1.c Library Termination Support

Classification: Basic

Requirement:

Library termination support - The system facilities required to terminate the use of a PSL must be provided. Such termination is defined to include the deletion of all library data storage and related indices.

Purpose of Paragraph:

To specify the need for facilities to support PSL termination and to define the nature of such termination.

Existing Tools which Satisfy this Requirement:

Any utility program or system command which can erase files will fulfill this requirement. Typical utilities used in small computer systems are FMP for Hewlett-Packard's RTE operating system series, and PIP for DEC's RT-11. Larger systems such as CDC or IBM tend to incorporate file deletion as a user-prompted system function.

Examples:

Backup of files prior to termination is always advisable. The methods to do this have been illustrated in the examples associated with SPS V Paragraph No. 3.1.1.c.1. The remaining facets of file, library and/or project termination involve deleting file/library/project references in the appropriate catalogues and actually deleting the files. The former is accomplished by editing the catalogues. Compaction of a file after editing is automatic. The methods used to perform file deletion are shown below.

Example 3.4.1.c.1: Deletion of files on a Type I system.

In the DEC system files may be deleted either individually or in groups of files with the same file name or extension name. The latter is shown in Figure 3.4.1.c-1, where all programs that were part of project 5, subdivision 5 are deleted. If individual file names are given, up to six files may be in-

cluded in a single deletion. For example, the deletion command could have been:

B55RAN.FOR, B55RAN.SAV, B55SYS.TXT, B55SYS.FOR, B55SYS.OBJ/D

Note that the command may not exceed a single terminal input line.

Similar rules apply to the PURGE command, which is the Hewlett-Packard counterpart of the "/D".

```
.R PIP
*DK: A/C
24-MAR-73
B55RAN.FOR      4 24-MAR-73
B55RAN.SAV     13 24-MAR-73
B55SYS.TXT      2 24-MAR-73
B55SYS.FOR      6 24-MAR-73
B55OPR.SAV     31 24-MAR-73
B55OPR.FOR      6 24-MAR-73
B55SYS.OBJ     14 24-MAR-73
B55SYS.SAV     20 24-MAR-73
3 FILES, 100 BLOCKS
3212 FREE BLOCKS
*B55RAN.*/D
*B55SYS.*/D
*DK: A/C
24-MAR-73
B55OPR.SAV     31 24-MAR-73
B55OPR.FOR      6 24-MAR-73
2 FILES, 37 BLOCKS
3275 FREE BLOCKS
*IC
```

Figure 3.4.1.c-1. Deletion of files in a Type I system.

Example 3.4.1.c.2: Deletion of files in a Type II system.

File deletion in the commercial time-sharing systems examined is very similar to that of the Type I systems (Figure 3.4.1.c-2). Note that an imbedded asterisk may be used to obtain file listings, but it may not be used for file deletion. While this may seem an inconvenience, the feature helps to prevent accidental file erasure.

```
15.03.27 >listf b55* * P (noitem
FILENAME FILETYPE MODE
B55RANDM FORTRAN    P
B55SYSTM FORTRAN    P
B55RANDM MODULE     P
B55SYSTM MODULE     P
B55SYSTM SCRIPT     P
B55SYSTM TEXT       P
```

```
15.03.43 >erase b55randm * P
```

```
15.04.16 >erase b55systm * P
```

```
15.04.28 >listf b55* * P (noitem
FILE NOT FOUND
!!E(00002)!!
```

```
15.04.43 >
```

Figure 3.4.1.c-2. Deletion of files in a Type II system.

Example 3.4.1.c.3: Deletion of files in a Type III system.

Deletion of files in the SCOPE system is dependent upon the original mode of storage. Since SCOPE's interactive system INTERCOM supports all SCOPE commands as well as its own commands, a file may be made permanent via either the STORE or CATALOG command. If the former was used, it may be deleted with the DISCARD command, as is shown below in Figure 3.4.1.c-3. If the latter command was used for file storage, then the PURGE function is used.

The file may be deleted in either case with or without first attaching it, i.e., making it a local file. If the deletion command is given without first attaching the file, then additional parameters are needed to identify the file. Note that each file must be deleted individually.

```
COMMAND-    FETCH, BILL55RANDOMFORTRAN, GAERTNER
COMMAND-    DISCARD, BILL55RANDOMFORTRAN.
COMMAND-    DISCARD, BILL55RANDOMLOADMOD, GAERTNER
COMMAND-    PURGE (BILL55SYSTEMLOADMOD, GAERTNER,
COMMAND-    PURGE(S)                                CY=3)
```

Figure 3.4.1.c-3. Deletion of files in a Type III system.

3.4.2 Full Library System Maintenance Requirements

Requirement:

Full Library System Maintenance requirements include all of the Basic Library System Maintenance requirements and the following additional requirements.

Purpose of Paragraph:

To introduce the statement of the Full Library System Maintenance requirements.

3.4.2.a Full Library Installation Support

Classification: Full

Requirement:

Full library installation support - The PSL installation procedures and related system support facilities must provide for the following two installation requirements.

1. Definition of the hardware configuration at installation time to meet the user's requirements (e.g., no specific hardware configuration beyond general minimum requirements).
2. A system generation facility that allows the user to generate a PSL that contains only the major functional capabilities (e.g., management data collection and reporting, etc.) required at that installation. This capability provides the flexibility to generate a PSL to meet requirements that fall in between the Basic and Full requirements as defined in this report.

Purpose of Paragraph:

To state that specification of both hardware and software current status and specification of minimum hardware and software desired is necessary for Full implementation of a Library System Maintenance facility.

Discussion and Examples:

The main intent of this paragraph is to ensure that the user is clear as to what hardware and software he needs to fulfill the functions of his installation and how his current status compares to those needs. Ideally, the hardware and software facilities at a given installation are no more and no less than exactly what is needed. This, of course, cannot always be the case, since there is bound to be a lead time (sometimes of more than a year) for acquiring new system components. Unless a data processing manager is extremely perspicacious and fortunate, this exact correlation between needs and current status is impossible. Moreover, an exact needs definition is not always possible if the computer system is used by several often unrelated groups.

Despite the foregoing considerations, it is still possible for the system manager to maintain a hardware and software needs definition in the external PSL. This need be no more than a statement of system component, why it's required, and when it was (or will be) obtained. Since the needs of a data processing installation are usually not static, this list should be intermittently reviewed and updated.

Not all projects undertaken by a contractor will require the same facilities. In terms of software support this may prove confusing unless a listing is maintained as to which catalogues, indices and utilities are being used for a given project. Such a listing might be termed a PSL Definition File, and could either be incorporated in the project catalogue or be kept separately. Since the Government supplied software to implement a PSL will be modular in nature, the contractor may acquire the modules as needed. He may also use them only as needed. For example, if there were one particular project which did not require any security, those indices and index maintenance programs which supported software security could be ignored. They would not be listed in the PSL Definition File, and the project librarian would not bother with them.

3.4.3 Summary of Facilities for Library System Maintenance

At present, much of the software required for even the Basic implementation of an LSM system does not exist in the strictest sense, though many existing file systems come very close to meeting the requirement.

Fortunately, a coherent set of programs is not vitally necessary to maintain the system library. All of the three system types examined allow a PSL to be established at a Basic level. File space may be allocated, altered, and unassigned as necessary. Ideally there would be a set of programs to automate this function, but human labor can still be used until the time that the software is available.

The PSL Catalogue can be established with the editor in the format shown on page 5-6 of SPS Volume VI. The Project Catalogue location information could be expanded to include file designation. File designation could be substituted for the DASD location, though it is best to have information on the physical location of data. When a new project is started the PSL Catalogue file could be edited to update it.

The Project Catalogue for each new project would also be created via an editor. If all programmers keep a daily log of all file modules generated and what project they relate to, these logs can be handed to a project librarian who will enter the information into the Project Catalogue to keep it up-to-date. The Project Catalogue format would remain essentially as described on p. 5-8 of SPS Volume VI. However, since not all programs dealing with a given aspect of a project can usually be stored contiguously, it might be necessary to merge the Project Catalogue and part of the Library File Index into one file.

The word "file" in the SPS report denotes a collection of "units" (termed "files" here) of the same type, e.g., all files that are object modules. Therefore, what the SPS describes as the File Accounting Record portion of the Library File Index is accounting data pertaining to all programming units of the same type. That organization implies passwords and security classification, etc., are dependent solely upon the type of data contained within the unit. This may not always be true. Individual installations may want to filter file access at the project subdivision level or the particular file level, rather than at the file type level. It seems in keeping with the concept of a PSL to allow such flexibility,

which means that the catalogue format cannot be fixed. For example, pointers to accounting records may be affixed to any of three levels of file identification.

The Project Catalogue as shown in Figure 3.4.3-1 assumes that the accounting information given in Table 5-2 of SPS Volume VI (file password, security classification, data compression, etc.) is applied at the project subdivision level. It is further assumed that file name designations adhere to the rules suggested in this report's examples associated with SPS V Section 3.2.2.d. Given the nature of the majority of operating systems commercially used, it may be difficult to obtain or keep current the DASD addresses of various files. If such is the case, the complete file designation (i.e., the information that is stored on a disk directory) may be used rather than the address to locate a file. Thus the "File DASD Address" column could be eliminated. The Account Record Pointers (there is one for project subdivision and one for file) may be either a pointer to some place within the same file or the name or address of another file which contains the desired information.

The Account Record for the subdivision would be as shown in SPS VI, Table 5-2. The Account Record for an individual file would contain the data illustrated in SPS VI Table 5-3, except that the starred items would not necessarily be automatically collected.

In order to implement a Library System Maintenance program, a contractor must either have a full time librarian and well codified documentation procedures for the programmer, or have a part time librarian, moderately good documentation standards, and a good set of programs to automate the book-keeping. The former option will probably be chosen until the software for the latter option has been developed.

AAAAAAA*		Project Name		
Project Subdiv. Code Subdiv. 1		Project Subdiv. Purpose, S1	Project Subdiv. Acct. Rec. Ptr., S1	
.		.	.	
.		.	.	
.		.	.	
Project Subdiv. Code Subdiv. n		Project Subdiv. Purpose, Sn	Project Subdiv. Acct. Rec. Ptr., S1	
File Name Subdiv. 1	File Type	File Vol. ID	File DASD Address	Ptr. to File Acct. Rec.
.
.
.
.
File Name Subdiv. n	File Type	File Vol. ID	File DASD Address	Ptr. to File Acct. Rec.

Figure 3.4.3-1. Suggested Project Catalogue Format

3.5 Data Security

Requirement:

This functional area involves control over the integrity and security of the data stored in the library. The PSL must provide control over different versions of programs/systems under development, control over the inadvertent destruction of data, and the protection of data from unauthorized access. The requirements in the area of data protection provide data privacy and support the protection of classified data. However, these are minimum capabilities and are not intended to provide a completely secure system. Protection of classified data is left primarily to physical security measures.

Purpose of Paragraph:

To define the Data Security functional area and to affirm that software data protection must be reinforced with physical security systems in order to protect classified data.

3.5.1 Basic Data Security Requirements

Classification: Basic

Requirement:

The ability to recover from inadvertent loss or destruction of data through the use of a backup capability is covered in SPS V, Paragraph 3.1.1.c.

In addition to such backup capability, the PSL must also provide, as a Basic requirement, protection against the inadvertent destruction of data as a result of an updating operation. The PSL must prevent the addition or copying of files into a PSL library if units with the same names already exist within that library.

Purpose of Paragraph:

To specify the need to prevent loss of data during an update operation.

Examples:

All known systems offer the capability of renaming files and making copies of files. Those capabilities when used in conjunction with well-defined installation procedures may be used to prevent data loss from update operations. It is strongly suggested that each installation, particularly those without automatic safeguards against data loss, establish a procedure whereby all personnel who deal with computer files adhere to the following rules:

- 1) Prior to beginning an update activity, personnel should note on a designated form the names of all files they will be updating or creating.
- 2) Personnel should determine if files already exist by that name.
- 3) If an update operation is involved, then a same-name file will already exist. That file should be renamed according to some installation-established convention (e.g., PROJCT.TXT could be renamed PROJCT.OTX to stand for "old text"). A copy of the file can then be made which, after update, will become the current file.
- 4) If personnel are creating a file and find that one already exists with the same name, then the name of the new (as yet uncreated) file should be altered.

Procedures such as the above may not only eliminate data loss due to update, but also serve to catch a variety of procedural errors such as duplication of work. Well defined conventions for naming and renaming of files should be established and vigorously adhered to, otherwise the above rules cannot act as safeguards.

The update activity form might appear as in Figure 3.5-1.

Name		Date		
Project		Subdivision		
Filename	Activity	Already Exist?	New	
	Update New		Filename	

Figure 3.5-1. Sample update activity log.

Example 3.5.1.1: Recovery from and prevention of loss or destruction of data in a Type I system.

Recovery from data loss by means of magnetic tape backup was demonstrated in Example 3.1.1.c.1 in Chapter 2 of this report. Such functions are normally handled by utility programs in Type I systems; DEC uses the Peripheral Interchange Program while Hewlett-Packard uses the Disk Backup programs.

Utility programs (PIP for DEC and FMP for Hewlett-Packard) also offer the capabilities of changing file names and of copying files disk-to-disk. The editor for RT-11 further offers an automatic backup facility when a file is edited with "EB".

The dialogue 3.5.1-1 shows how the editor and PIP may be used to prevent data loss in update operations. First, the user requests a listing of all files. Assuming the user's intent is to update the source code for J39DRV, it is necessary to rename (or delete if desired) the already existing backup file. Otherwise it would be destroyed at the end of the editing session in which the source was accessed with the EB request. Another file listing is made following the edit session to demonstrate how the files have been renamed.

If the revised source code is to be compiled, loaded, executed, and the documentation updated, then old files dealing with those operations will have to be saved. Again the rename switch ("/R") is used to give a new name to the already existing load module, J39DRV.SAV. This clears the way for the automatic creation of the next .SAV extension during loading. Since the program documentation (.TXT extension) is not automatically generated, the old file can be renamed and copied all in one step, allowing the user to update the text file. Note that the file names of the program and the documentation are different. This is necessary so that Edit Backup sessions

on the text file do not destroy backup of the source file. This feature would also allow the user to rename the newly created J39DVR.BAK to J39DVR.OTX after the editing session, rather than make a copy beforehand.

```
.R PIP
*/
4-MAR-73
J39DRV.FOR 15 2-MAR-73
J39DRV.BAK 12 2-MAR-73
J39DRV.SAV 13 3-MAR-73
J39DVR.TXT 4 1-MAR-73
4 FILES, 44 BLOCKS
3173 FREE BLOCKS
*J39DRV.BAK<J39DRV.BAK/R
*IC
```

```
.R EDIT
*EBJ39DRV.FOR$R$$
*
```

. . .

```
*$E$$$
```

```
.R PIP
*/
4-MAR-73
J39DRV.BAK 15 2-MAR-73
J39DRV.BAK 12 2-MAR-73
J39DRV.SAV 13 3-MAR-73
J39DVR.TXT 4 1-MAR-73
J39DRV.FOR 16 4-MAR-73
5 FILES, 60 BLOCKS
3132 FREE BLOCKS
*J39DRV.SV1<J39DRV.SAV/R
*J39DVR.OTX<J39DVR.TXT/A
*/
4-MAR-73
J39DRV.BAK 15 2-MAR-73
J39DRV.BAK 12 2-MAR-73
J39DRV.SV1 13 3-MAR-73
J39DVR.TXT 4 1-MAR-73
J39DRV.FOR 16 4-MAR-73
J39DVR.OTX 4 4-MAR-73
6 FILES, 64 BLOCKS
3173 FREE BLOCKS
*IC
```

Figure 3.5.1-1. Recovery from and prevention of loss or destruction of data in a Type I system.

AD-A081 389

BAERTNER (W W) RESEARCH INC STAMFORD CONN
PROGRAMMING SUPPORT LIBRARY, VOLUME II. GUIDELINES FOR IMPLEMENTATION--ETC(U)
NOV 79 C M TURCIO, W M SCHREYER, N A ADAMS F30602-78-C-0103

F/G 9/2

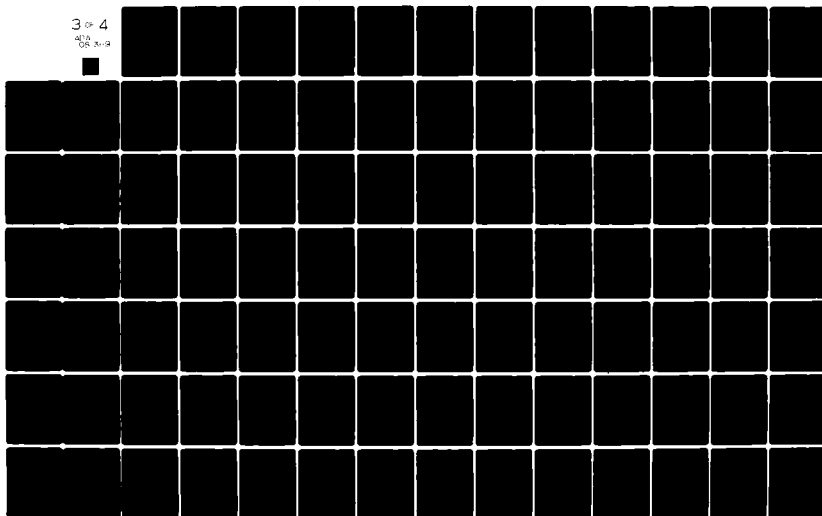
UNCLASSIFIED

RADC-TR-79-241-VOL-2

NL

3 of 4

412
OR N-9



Example 3.5.1.2: Recovery from and prevention of loss or destruction of data in a Type II system.

Example 3.1.1.c.2 in Chapter 2 illustrated recovery from data loss via magnetic tape backup. Just as backup operations are easily performed in these time-sharing systems, so loss prevention activities are also facilitated. Both CSS and CMS offer three alternative methods: File copying and renaming, renaming the output file upon termination of editing, or automatic renaming and generation of an update log if the UPDATE function is used. The dialogue in Figure 3.5.1-2 demonstrates the use of these methods.

After a file listing is obtained, an update file is created which contains control statements specifying update operations on certain records within a source file, as well as sequencing characteristics for the new file. When the UPDATE function is called with the name of the source file, that file is then updated according to the UPDATE file with the same name. Two new files are created: A log of update functions performed and any resulting error messages (file type UPDLOG), and the updated file itself. The latter has a file name beginning with a "." followed by the first seven letters of the source file name. The existence of these files is verified by the listing.

Unfortunately, the naming system used by UPDATE is not necessarily in agreement with installation standards, so renaming of files is necessary. The load module for the previous version is also renamed so it can be saved. Finally, the documentation J39DVR1 SCRIPT is edited for textual updating, and the new version filed away as J39DRVR2 SCRIPT.

```

17.52.14 >listf J39* * P (noitem
FILENAME FILETYPE MODE
J39DRIVR SRCBAKUP P
J39DRIVR MODULE P
J39DRVR1 SCRIPT P
J39DRIVR FORTRAN P

```

```

17.52.22 >edit J39drivr update
NEW FILE.

```

```

INPUT:

```

```

>./ s 10 10
>./ i 50
>      x = y ** 3
>./ r 90
>      z = a(n)
>./ d 120 140
>

```

```

EDIT:

```

```

>file

```

```

17.53.34 >update J39drivr fortran

```

```

17.55.07 >listf *39* * P (noitem

```

```

FILENAME FILETYPE MODE

```

```

J39DRIVR UPDATE P
J39DRIVR SRCBAKUP P
J39DRIVR MODULE P
J39DRVR1 SCRIPT P
J39DRIVR UPDLOG P
J39DRIVR FORTRAN P
.J39DRIV FORTRAN P

```

```

17.55.28 >alter J39drivr srcbakup P J39drivr srcbak1 P

```

```

17.57.11 >alter J39drivr fortran P J39drivr srcbak2 P

```

```

17.57.57 >alter .J39drivr fortran P J39drivr fortran P

```

```

17.58.28 >alter J39drivr module P J39drivr load1 P

```

```

17.59.25 >edit J39drvr1 script P

```

```

EDIT: ...

```

```

>file J39drvr2 script P

```

```

FILING: J39DRVR2 SCRIPT P.

```

```

18.00.25 >listf J39* * P (noitem

```

```

FILENAME FILETYPE MODE

```

```

J39DRIVR UPDATE P
J39DRIVR SRCBAK1 P
J39DRIVR LOAD1 P
J39DRVR1 SCRIPT P
J39DRVR2 SCRIPT P
J39DRIVR UPDLOG P
J39DRIVR SRCBAK2 P
J39DRIVR FORTRAN P

```

Figure 3.5.1-2. Recovery from and prevention of loss or destruction of data in a Type II system.

Example 3.5.1.3: Recovery from and prevention of loss or destruction of data in a Type III system.

Magnetic tape backup as a means of recovery from data loss was covered for Type III systems in Chapter 2, Example 3.1.1.c.3. Therefore the discussion here will be limited to data loss prevention.

The CDC Scope system used for this example automatically allows five different versions of the same file to coexist. Each version has a separate cycle number, which is assigned when the file is created and incremented each time the file is catalogued again. Note that the use of the interactive STORE command does not update the cycle number.

The dialogue in Figure 3.5.1-3 shows how a user may obtain a listing of his permanent files via the AUDIT and PAGE commands prior to updating a source file. In order to edit that file the user must first ATTACH it to make it local. When the file is later CATALOGed, a new cycle number is assigned as shown.

The last five program versions are retained. When cycle number 6 is assigned, cycle 1 is automatically erased. Cycle numbers may go as high as 999, after which they must be reset with the RENAME command.

Additional data loss protection may be given by making differently named copies of a file, or, if appropriate, by renaming a file. These functions are accomplished by the COPY and RENAME commands respectively. It should be noted that the latter command requires that the file first be attached as a local file.

COMMAND- AUDIT(LF=FILELIST,ID=GAERTNER A1=P)
 COMMAND- PAGE,FILELIST

AUDIT OF 795 PERMANENT FILES PARTIAL ID TIME
 17.05.17 03/02/79 PAGE NO. 1

SETNAME=SYSTEM

OWNER	PERMANENT FILE NAME	CYCLE	ACCOUNT
UNIT	PROS CREATION EXPIRATION LAST ATT LAST ALT		
GAERTNER	J39DRIVERFORTRAN	1	GRT 451
0050	12 07/10/79 06/30/79 02/17/79 02/17/79		
GAERTNER	J39DRIVERFORTRAN	2	GRT 451
0051	14 02/17/79 06/30/79 02/21/79 02/21/79		
GAERTNER	J39DRIVERLOADMOD	1	GRT 451
0050	13 02/25/79 06/30/79 02/25/79 02/25/79		
GAERTNER	J39DRIVERDOCUMENT1	1	GRT 451
0053	3 01/19/79 06/30/79 02/10/79 02/01/79		

COMMAND- ATTACH,S,J39DRIVERFORTRAN, ID=GAERTNER

PF CYCLE NO.=002
 COMMAND- EDITOR

..S,X,N
 ..BYE

COMMAND- CATALOG,X,J39DRIVERFORTRAN, ID=GAERTNER
 NEW CYCLE CATALOG
 RP= 090 DAYS
 CT ID= GAERTNER PFN=J39DRIVERFORTRAN
 CT CY= 003 0000392 WORDS.:

COMMAND- ATTACH(Y,J39DRIVERLOADMOD, ID=GAERTNER)
 COMMAND- RENAME(Y,J39DRIVERLOADMOD1, ID=GAERTNER)
 COMMAND- COPY(J39DRIVERDOCUMENT1,J39DRIVERDOCUMENT2)
 COMMAND- AUDIT(LF=FILELIST2, ID=GAERTNER, AL=P)
 COMMAND- PAGE,FILELIST2

AUDIT OF 197 PERMANENT FILES PARTIAL ID TIME
 17.05.17 03/02/79 PAGE NO. 1

SETNAME=SYSTEM

OWNER	PERMANENT FILE NAME	CYCLE	ACCOUNT
UNIT	PRUS CREATION EXPIRATION LAST ATT LAST ALT		
GAERTNER	J39DRIVER FORTRAN	1	GRT 451

Figure 3.5.1-3. Recovery from and prevention of loss or destruction of data in a Type III system.

0050	12	01/10/79	06/30/79	02/17/79	02/17/79	
GAERTNER		J39DRIVERFORTRAN			2	GRT 451
0051	14	02/17/79	06/30/79	02/21/79	02/21/79	
GAERTNER		J39DRIVERFORTRAN			3	GRT 451
0052	19	03/02/79	06/30/79	03/02/79	03/02/79	
GAERTNER		J39DRIVERLOADMOD1			1	GRT 451
0050	13	02/25/79	06/30/79	02/25/79	02/25/79	
GAERTNER		J39DRIVERDOCUMENT1			1	GRT 451
0051	3	01/19/79	06/30/79	02/10/79	02/01/79	
GAERTNER		J39DRIVERDOCUMENT2			1	GRT 451
0052	3	03/02/79	06/30/79	03/02/79	03/02/79	

Figure 3.5.1-3 (continued). Recovery from and prevention of loss or destruction of data in a Type III system.

3.5.2 Full Data Security Requirements

Requirement:

In addition to the Basic Data Security requirements a PSL must support the following requirements in order to provide control over the integrity and security of the data stored in the library.

Purpose of Paragraph:

To introduce the Full requirements for Data Security.

3.5.2.a Full Data Integrity

Classification: Full

Requirement:

Data Integrity - A method for maintaining control over multiple versions of a program or program system as well as a method for maintaining control over various change levels of source files within a specific version of a program system must be provided. The objective of such control is to allow a development and/or maintenance group to manipulate multiple versions of the same program system (which would be stored in separate subdivisions) while still ensuring that source code is not duplicated unnecessarily. Such control also allows data to be merged from several subdivisions in order to obtain the desired combination for the purpose of performing tests (see also Paragraph 3.9.2.a).

It is to be noted that a version is defined as a complete program or program system as it exists at a specific point in time, usually related to a certain phase of development. Any given program system might have an operational version, a development version and a maintenance version, among others. A change level within a version refers to the number of major changes that have been made to the individual source files within the version.

Purpose of Paragraph:

To emphasize the need to maintain data integrity by means of control over the variations within a programming system as they exist over a period of time.

Examples:

Control over program versions and change levels is primarily affected by means of naming conventions, though internal program documentation in the form of comments may also serve to designate version and change level. All three types of computer systems allow the establishment of such naming conventions. Specific examples follow.

Example 3.5.2.a.1: Data integrity control in a Type I system.

The suggested naming conventions developed thus far for six character file names allow one character each for programmer, project, and project subdivision, and three characters for the file name. If version designation is also to be included, then the file name indicator can be reduced to two characters, and the last character used to represent the version. For example, a random number generation program assigned to programmer B as part of project 5, subdivision 5 might have as the name for the FORTRAN source of the maintenance version "B55RNM.FOR".

It is suggested that change level of source code be indicated by comments preceding the actual source code. Such comments should include a numerical designation for the change level, the name of the programmer who made the changes, what the changes were and why they were made, and the date of change. The entire chronological history of changes should thus be recorded at the beginning of each program. This textual data may be parsed in order to generate management information. It is further suggested that source listings of change levels for at least the last three levels be maintained in the external library.

It is assumed that the object and load modules for a given program or program set of a specific version will be associated with the latest change level. Installation procedures should include practices which validate that assumption, since it is critical for the accurate creation of merged program versions used for testing.

The merging of versions is implemented in various ways depending on the system used. Hewlett-Packard RTE-IV allows a leader command file to be built which names all files which are to be included in the program system loaded for execution. Execution of the loader then references this command file.

DEC's RT-11 utilizes an even simpler method, as shown in Figure 3.5.2.a-1. That figure illustrates how a development version of a system (B55VR4) may be built from a new version of program B and the operational versions of programs A and C. After the linker has been called, a command string is given which consists of the name of the linked file, the assignment operator, and all object modules to be linked. Here "V3" is used to indicate the operational version and "V4" the development version. The "/F" at the end of the command string instructs the Linker to use the FORTRAN library.

```
•R LINK
*B55VR4=B55AV3,B55BV4,B55CV3/F
*
```

Figure 3.5.2.a-1. Data integrity between versions of a system on a Type I system.

Example 3.5.2.a.2: Data integrity control in a Type II system.

The prime means of maintaining version integrity in this type of system is either by placing different version types on different storage devices, by designating a character of the file name to represent the version, or by assigning separate user identifications to different versions. Since commercial time-sharing systems usually allow the user to have various virtual disks attached to his machine, he may allocate one disk to storage of developmental versions, another to production versions, etc. However, this method may cause problems if someone inadvertently transfers a file between disks. Therefore, it is advised to either encode the version designation into the file name or, if possible, to have an entirely separate user identification (i.e.,

virtual machine) for each version. The latter method would provide the added data protection by screening data access. For example, the production version could be accessed only in a read-only manner by all machines other than the one with the production identification.

Change level integrity may be maintained by well-organized commenting procedures within source files. A sample of such procedures was given in Example 3.5.2.a.1.

Linkage of different versions to create a new version for test purposes is accomplished simply by specifying the names of the object modules to be linked together. If file type and file mode (disk designation) are not given, a file type of TEXT is assumed and all attached disks are searched for the named object module. If different versions are stored under separate user ID's, the main disk of the other virtual machine(s) may be attached on a temporary basis, thus expanding the scope of search for the named object modules. In Figure 3.5.2.a-2 below, the main (191) disk of user ID PRODUCT is attached as auxiliary disk, address 192, in a read only mode. When the load command is issued all disks including the 192 T disk are searched. Normal system response (the time and ">") indicates successful loading. The GENMOD command produces a file which has a filename the same as the first named file in the LOAD command and a filetype of MODULE.

```
15.39.42 >ATTACH PRODUCT 191 AS 192 RO
PRODUCT ATTACHED AS 192,(RO)
```

```
15.39.53 >LOAD A B C
```

```
15.40.33 >GENMOD
```

```
15.42.15 >
```

Figure 3.5.2.a-2. Example of preserving data integrity between development and production versions of a program on a Type II system.

Example 3.5.2.a.3: Data integrity control in a Type III system.

This more limited system allows only naming conventions to reflect version designation, although the cycle numbers (reference Example 3.5.1.3) may be used in addition to internal comments to designate change level of source code. The SCOPE system fortunately permits a maximum 40 character

file identifier, which is more than adequate to contain all the needed information (programmer, project, subdivision, file type, file name, and version).

When it is necessary to link together modules from separate versions, the same procedure is used as linkage for modules of the same version. Figure 3.5.2.a-3 illustrates how the three files are attached then all copied into a single file (S) prior to executing the loader with that file as input. For further information on loader operation please reference Example 3.3.1.b.3.

COMMAND- ATTACH, A, APRODUCTION, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- ATTACH, B, BDEVELOP, ID=GAERTNER

PF CYCLE NO.= 003

COMMAND- ATTACH, C, CPRODUCTION, ID=GAERTNER

PF CYCLE NO.= 001

COMMAND- COPYBR, A, S, 99

END OF FILE ENCOUNTERED AFTER COPY OF
RECORD 3

COMMAND- COPYBR, B, S, 99

END OF FILE ENCOUNTERED AFTER COPY OF
RECORD 2

COMMAND- COPYBR, C, S, 99

END OF FILE ENCOUNTERED AFTER COPY OF
RECORD 10

COMMAND- XEQ, LOAD=S, NOGO

Figure 3.5.2.a-3. Example of preserving data integrity between program versions on a Type III system.

3.5.2.b Full Data Protection

Classification: Full

Requirement:

Full Data Protection - A Full implementation of a PSL must include the protection of data from unauthorized access and unauthorized update. The data protection requirements are:

1. The PSL must provide a means to restrict access to the subdivisions within a project and the data types within a subdivision.
2. The PSL must provide a means to filter access to files of a given data type stored within a subdivision. Specifically, it must be possible to:
 - (a) Restrict access to data files for either output or update to only authorized individuals.
 - (b) Allow free access to data files for output but restrict update to only authorized individuals.
 - (c) Allow free access to data files for either output or update.
3. The PSL must provide a means of printing classified titles (i.e., Confidential, Secret, etc.) on any printed reports produced by the system.

Purpose of Paragraph:

To emphasize the need to limit data access to authorized individuals. Such limitations must be able to be implemented for all levels of PSL organization and for all access functions.

Examples:

Example 3.5.2.b.1: Data Protection in a Type I system.

Type I systems are designed to be fairly small. RT-11, for example, is a single user system and as such there is no need for data security measures. If an RT-11 user wishes to secure his data, the only means available is to store the data on a removable device (e.g., tape cassette) and to physically secure that device.

However, there are other operating systems that may be used on PDP-11s which are meant for multi-user environments. RSTS/E, RSX, or any of the variations in the latter such as TRAX and IAS all support data security features. Device initialization procedures on RSTS/E include establishment of volume protection which would limit access to project subdivisions, assuming each subdivision is stored in a separate volume. That operating system also associates an eight bit protection code with each file. The code is used to designate access level and may be set to allow the three access types specified in sub-paragraphs (a), (b), and (c). Since it is a multi-user system, RSTS/E also has an accounting procedure which can be used to limit file and volume access by account name.

The Hewlett-Packard RTE-IV operating system and its earlier versions, RTE-II and -III are all designed to be multi-user. While there is no volume protection software available, each file has a code associated with it that designates its protection level: 0 for no protection, a positive number for read only protection, and a negative number for full protection.

None of the Type I systems surveyed provide a means for printing classified titles on reports other than editing the report print file to include such title. This of course cannot be done with all types of print file. Additional software would be required.

Example 3.5.2.b.2: Data Protection in a Type II system.

Commercial time-sharing systems by their very nature require strict data protection, therefore are designed to provide extensive protection facilities. Both systems surveyed have initial user filtering procedures. The user must know a valid user ID and the correct password associated with it. CSS further requires specific knowledge of the designation of the host computer used to house the files. After such information is given and logon successfully accomplished, a special file, PROFILE EXEC, is executed if it is resident on one of the disks attached to that user's virtual machine. That file may require the user to give another password, or more than one, in order to stay logged onto the system. CSS also has another similar automatically executed file, PROTECT EXEC. This file can restrict user access to any or all CSS commands and facilities as well as to user programs and data files, thus limiting full range commands to the one user who is owner of the disk.

Both systems use passwords for reading and writing privileges in disk linkages. Since this is a shared resource type of system, disks may be requested and attached to a user's machine; they may be the permanent disk of another user or a mountable disk. This necessitates a method for data protection. CSS utilizes a directory which contains information on all users in the system, the disks they access and the associated passwords. This information is established when a user first enters the CSS system. The passwords for read and write disk access are required whenever a user requests the disk via the ATTACH command, and they may be set so as to allow read, write or update privileges to all users, to no one except the owner, or to authorized users only. Furthermore, disks may be made proprietary, which results in their being automatically detached whenever a user initializes or reinitializes the CSS system.

Figure 3.5.2.b-1 shows a CSS session in which the user must specify correct passwords to gain disk access. First, the link to the host computer is established and the user ID given. If the latter is not correct linking must begin all over again. The user then has two chances to give a correct password before he is forced off the system and has to begin again at the link step. In order to attach a user's disk, that disk must be attachable and the required passwords given correctly. Finally, a PROFILE EXEC or PROTECT EXEC may request additional passwords.

The data protection facilities in CMS are not quite as thorough or flexible. Each disk has associated with it read and write passwords which must be issued when the disk is attached, but there is no gradation of access privileges. These passwords may be used in conjunction with filemode levels, however, to filter individual file access.

While commercial time sharing facilities are not suitable for classified work due to the essential nature of such systems, both offer means for printing titles designated by the user. CSS utilizes the TITLE command while CMS requires user purchase of the SCRIPT text processor to implement title printing. In both cases, however, the user must designate the title to be printed. It is not automatically generated by the system. Further software could be required to automate this feature.

```

CSS ONLINE - HSYS

>link hsys

>login Jones
invalid id 'Jones'.
LINK

>link hsys

>login adams
password:
*****
password incorrect.
password:
*****
A/C INFO:
>demo session
HSYS READY AT 15.46.38 ON 20APR79.
CSS.302 01MAY78

15.46.47 >attach smith * as t wr
PASSWORD:
*****
BAD PASSWORD
!!E(00005)!!

15.47.05 >attach smith * as t wr
PASSWORD:
*****
SMITH ATTACHED AS T-DISK

PLEASE ENTER SECONDARY PASSWORD FOR SMITH :
*****

```

Figure 3.5.2.b-1. Data protection in a Type II system.

Example 3.5.2.b.3: Data Protection in a Type III System.

The SCOPE operating system used on CDC 6600 computers was designed primarily for batch operations although it does have an interactive mode. Both batch and interactive facets were designed to allow tight data security, which is one reason why this system has been so heavily used at Government installations.

All permanent files created in the SCOPE system have an associated four-bit permission code which indicates allowed file access. This code and related passwords are set

when the file is initially catalogued. Valid access levels are "read only", "modify" (like update), "extend" which allows the file size to be expanded, and "control" which is needed to add or delete a file. The user may further define a fifth password which is used to validate the other four. These passwords, if defined, must be given in order to attach, purge, or rename a file. Since this system is completely file-oriented, the password set serves to restrict user access on all organizational levels, both project and subdivision.

The INTERCOM interactive subsystem of SCOPE further requires that a user give his ID and associated passwords before he can logon. Passwords at this level are classed as restricted and unrestricted, the latter being intended for classroom and demonstration use while the former is specific to a given user. This password also is associated with the user's permanent password file wherein maximum field length and time limit are defined, as well as the level of commands available to the user.

Figure 3.5.2.b-2 shows the use of passwords. Note that the login password is blacked out. A file is then created with the editor and catalogued specifying a retention period of 90 days, read password of X, control password of Y, modify password of A, and a turnkey password of C. MR=1 specifies that the file may have multi-user read only access. Attempts to attach the file without the proper passwords fail. Since the turnkey password was defined during cataloguing, it must be given. The file is attached with read, modify and extend access when all passwords are given, and the RW parameter set to a positive value to allow a single rewrite or extension.

Neither SCOPE nor its subsystem INTERCOM offer a means of printing titles on reports, other than through the text editor.

Requirements for Necessary New Software:

None of the systems considered offered facilities to automatically generate data security classification titles in printout. This would not be possible until each file had its own accounting file as suggested in SPS Volume VI, pages 4-6, 5-10 and 5-21. While the "Unit Accounting Record" is not initially designed to include file data security status, that could easily be added on. Once this is done, the output processor could be modified to call a program which would have the following specifications.

```

CONTROL DATA INTERCOM 4.1
DATE    02/07/79
TIME    10.07.25

LOGIN
ENTER USER NAME-   GAERTNER
XXXXXXXXXX PASSWORD

02/07/79  LOGGED IN AT 10.08.09
          WITH USER ID-   GAERTNER
          EQUIP/PORT  52/03
COMMAND-  EDITOR
          .
          .
          .
..SAVE,MYFILE
..BYE
COMMAND-  CATALOG,MYFILE,BILL55RJV1,
          ID=GAERTNER,RP=90,RD=X,CN=Y,
          MD=A,TK=C,MR=1
COMMAND-  ATTACH,S,BILL55FJV1,ID=GAERTNER
          ATTACH PERMISSION NOT GRANTED
COMMAND-  ATTACH,S,BILL 55RJV1,ID=GAERTNER,PW=
          Y, A, X
          ATTACH PERMISSION NOT GRANTED
COMMAND-  ATTACH,S,BILL55RJV1,ID=GAERTNER,
          PW=Y,X,C,A,RW=1,MR=1
PF CYCLE NO.=001
COMMAND-

```

Figure 3.5.2.b-2. Data protection on a Type III system.

Input:

- 1) The type of the item to be printed (e.g., compilation result, execution result, management report, etc.). This would probably require further system modification.
- 2) If item type was file specific, the name of the file would be needed as a second input parameter.

Program Performance:

- 1) For file specific printout, the accounting record for that file would be accessed to determine classified status. A switch could be set accordingly.
- 2) Other types of reports would each have a classified status associated with them. This information would be contained in a Print Status Table which would be accessed to determine the proper switch setting.
- 3) The switch setting determined by 1) or 2) above would indicate the title which would be inserted at the beginning of the material to be printed. This could be done in any number of ways, depending upon the output facilities of the system used.

Output:

Correct classified status title as heading of print-out.

3.5.3 Summary of Facilities for Data Security and Classified Work

All of the systems examined offered facilities sufficient to meet the Basic requirements of this functional area. Most also could fulfill the majority of Full requirements as well, with the exception of title printing. As was pointed out earlier, the RT-11 system was designed to be single user, so that data security features are not included. However, other operating systems which can run on the same hardware do offer data security measures of the same caliber as the other two types of systems.

Not all of these systems are suited to classified work. Type II systems, as mentioned previously, simply cannot be used for top security activities. Small in-house computers can be much more closely guarded, and so are a better choice for highly classified work.

The Type III system studied offered the best design features for classified data security. However, this system would have to be used in a limited fashion in order to obtain such security. Rather than remote terminal link up through telephone lines, classified work would have to be conducted onsite using a dedicated computer after the system had been cleaned of extraneous programs and the remote terminal lines disabled.

3.6 Management Data Collection and Reporting

Requirement:

Management Data Collection and Reporting - This function involves the collection and storage of data related to program development and maintenance and the generation of management reports containing the data and/or summaries of the data. For further details on data items collected, report classes, etc., see the Management Data Collection and Reporting report (Volume IX).

Purpose of Paragraph:

To introduce the Management Data Collection and Reporting functional area and to define the concerns of that area.

3.6.1 Basic Management Data Collection and Reporting Requirements

Classification: Basic

Requirement:

The collection and reporting of management data is not a Basic requirement of a PSL.

Purpose of Paragraph:

To state that this functional area is not a Basic PSL requirement.

3.6.2 Full Management Data Collection and Reporting Requirements

Requirement:

The functional requirements are subdivided into full collecting, full updating, full accumulating, full archiving and full reporting requirements. All management data collected by the PSL will enter the PSL through either the collecting or updating facilities.

Purpose of Paragraph:

To state the subdivisions of this functional area and to indicate the subdivisions responsible for obtaining data.

Discussion:

None of the systems investigated nor any other known existing system offers the data gathering and reporting facilities of the type required to fulfill the specifications of this functional area. In fact, the statement of these requirements implies that a PSL as described in Volumes V and VI of the SPS has been implemented, as there is no other way that the necessary data structures and related programs could already exist.

The key constraint upon the use of existing facilities to implement this functional area is the word "automatic". All the required data could be collected manually, but the cost of so doing would be prohibitive.

Therefore, the burden of compliance with the standards of this functional area must necessarily fall upon the Government, in that they should provide the prospective contractor with a complete PSL package which can suitably interface with that contractor's operating system. While the development of such a package and interfaces is an enormous task, it would be necessary in order to implement the Full version of the PSL.

3.6.2.a Full Collecting

Classification: Full

Requirement:

Full Collecting - The data collection facility is an automatic function of the PSL. This facility provides

- (a) for the gathering of Actual data, i.e., data describing the results of programming actions within the scope of a project. Such data is gathered when programming related data as defined in SPS V, para. 3.1.1.a is entered into the PSL.
- (b) for the storage of Actual data in a Management Statistical Data Storage area.

This facility must have the following capabilities:

1. Counting input source code lines, i.e., records, and storing or modifying already existing Actual data stored in the PSL. Such data may include version/modification level, net lines of source code per file, and total lines of source code input per file.
2. Gathering and storing in a Management Statistical Data Base all pertinent source file data (e.g., start date and end date) which is available to the PSL.
3. Counting and storing Actual data concerning certain PSL functions. Typical data would be the number of compilations, number of assemblies, number of lines of code from another source, etc.
4. Allowing the user to add routines to the PSL which may be used for the gathering of Actual data needed to satisfy unique user requirements.

Purpose of Paragraph:

To describe in detail the tasks allocated to the collection subdivision of the Data Management functional area.

Discussion and Examples:

None of the systems studied contain automatic data collection facilities. While the two commercial time sharing systems do count lines of source code, that data is not used for any comparative reports.

A similar problem arises with starting and ending dates since those systems which do not collect such data (e.g., the Type III SCOPE system, which uses the AUDIT command) do not meaningfully coordinate the information with other system data.

The following is a general discussion of implications of and problems relating to the specifications of this paragraph.

Subparagraph 1:

This specification assumes the existence of a Unit Accounting Record as detailed on page 5-21 of SPS Volume VI. The record described there would also have to include entries for net and total lines of source code.

There are two ambiguities contained in this subparagraph that must be resolved before designing the additional software to implement source code data collection.

- 1) Is it the goal to count lines (i.e., records) of source code, or to count source code statements? The assumption that the two are synonymous is not correct, particularly when dealing with languages designed and/or written according to structured programming principles.
- 2) How are "net" and "total" to be defined in terms of source code lines? Net may or may not include lines of documentation. It is assumed it does not include lines from another source.

Subparagraph 2:

This specification assumes the existence of both the Unit Accounting Record (SPS VI, p. 5-21) and the Management Data File (SPS VI, pp. 5-49 - 5-55), which together comprise the Management Statistical Data Base. The Unit Accounting Record as described there does not include unit ending date, which would have to input manually and would have to be estimated by the manager.

Subparagraph 3:

This specification assumes the existence of the Management Data File and of a PSL program COMPILE which interfaces with the user's operating system. This program would be necessary in order to monitor the number of compilations. Note that the outline for the COMPILE program (SPS Volume VI, pp. 5-64 - 5-66) fails to include statement of the source language.

Again, there are ambiguities contained in this specification.

- 1) Should the data herein specified be recorded at the Program Level of the Management Data File (MDF) or in the Unit Accounting Record? Table 5-3 of SPS VI shows the latter, yet data on compilation/assembly would seem to be appropriate to the program level as well. Furthermore, the term "Actual data" has consistently been used to refer to MDF entries, indicating inclusion of the data in the MDF. Note that this data is included in Program Production data items of the MDF as given in Appendix A of SPS Volume IX.
- 2) Does "lines of code from another source" refer to macros that may be stored independently or to sub-procedures that are to be included via an INCLUDE directive, or to both?

Subparagraph 4:

This specification implies the existence of an MDF which is expandable beyond what is described (SPS VI pp. 5-49 and 5-52 - 5-55). Alternately, it could be interpreted such that the Unit Accounting Record (UAR) is to be included in the class of Actual Data, even though that interpretation is not supported by SPS IX Section 3.5 on Data Classes and Types. Wherever the data resulting from additional routines is to be stored, the specifications for the program to add such routines is given on pages 5-62 - 5-63 of SPS Volume VI.

Requirements for Necessary New Support Software:

Initial software specifications are contained in Section 5.4 of SPS Volume VI. While these are incomplete, it is not possible to give more detailed specifications until the previously discussed ambiguities are clarified or unless certain assumptions are made. A tentative outline may be found in Section 3.6.3.

3.6.2.b Full Updating

Classification: Full

Requirement:

Full Updating - Provide a data update facility to add, delete, or replace Plan or Actual data stored in a Management Statistical Data Base. This facility must provide the following capabilities:

1. Allocating Management Statistical Data Storage area(s) on direct access devices and initializing the PSL for the data collection and reporting functions for a new structured programming project. These storage area(s) are reserved for data at both the hierarchical level (file, program, subsystem and system) and computer job level.
2. Editing Management Statistical Data and notifying the user of format or data errors.
3. Adding Management Statistical data, supplied by the user, to a Management Statistical Data Base in the PSL. This data will include such items as source file name, user identification, programming language, specification omission errors, implementation misinterpretation errors, and man-months or hours of programming time.
4. Deleting Management Statistical Data from a Management Statistical Data Base. Such deletion will be based on user-supplied information.
5. Replacing Management Statistical Data existing in a Management Statistical Data Base. Such replacement will be based on user-supplied information such as deliverable code indicator, number of lines of source code from another source, and man-months or hours of programmer time.
6. Storing user-supplied data relating to computer job turnaround time. This data will be used to compute the average, maximum and minimum computer turnaround time for a job.
7. Terminating the collection of data for specific data types (e.g., Computer Utilization). For clarification of data types see Section 3.5.1 of SPS Volume IX.
8. Terminating the data collection and archival functions performed by the PSL for a programming project.

Purpose of Paragraph:

To specify those tasks which are to be allocated to an update facility or program and which are necessary for the maintenance of the Management Statistical Data Base.

Existing Tools Which Satisfy this Requirement:

Editing, adding, deleting, and replacing data within the Management Statistical Data Base may all be accomplished by system-resident editors.

File manipulation utilities or commands such as Hewlett-Packard's FMP or DEC's PIP, may be used to allocate storage. Storage allocation tools were discussed previously in Chapter 3.4, Section 3.4.1.b.

Even though these tools exist, coordination of this paragraph's specifications with the MDF and other related data structures make it advisable to use a special-purpose Management Data Handling program such as the system suggested in Volume VI, Sections 5.4 and 6.4.

Requirements for Necessary New Support Software:

Initial program requirements are given in SPS Volume VI, Section 5.4 for batch implementation and Section 6.4 for on-line implementation. Those specifications are very high level and lack details as to actual program performance. A more specific presentation is given in Section 3.6.3 of this Chapter, wherein the entire set of programs needed to accomplish the tasks of Management Data Collection and Reporting is tentatively outlined.

3.6.2.c Full Accumulating

Classification: Full

Requirement:

Accumulating - Provide a data accumulation facility to extract and summarize or process Management Statistical Data stored in the PSL. This facility must provide the following capabilities:

1. Summarizing Management Statistical Data for the hierarchical levels of program, subsystem and system, and for all related source files existing in the PSL. The summarized data will include items such as total program source lines and total program files.

2. Processing Management Statistical Data which exists in the PSL. Such processing consists of determination of maxima, minima, and averages for certain items and may include the calculation of average source file size, average number of source lines input per file, and maximum number of updates allowed for a file.
3. Retaining the summarized and processed data for the purposes of reporting and archiving.

Purpose of Paragraph:

To define the types of derivative data which must be calculated on the basis of gathered data.

Discussion:

This paragraph presupposes the existence of a well-defined Management Statistical Data Base. The facilities to manipulate that data base and to further derive data from it do not currently exist, but are outlined briefly in SPS VI, Section 5.4. They are discussed more deeply in Section 3.6.3 of this Chapter.

3.6.2.d Full Archiving

Classification: Full

Requirement:

Full Archiving - Provide a historical information storing facility to record historical information on structured programming projects. This facility must provide the following capabilities:

1. Storing on a secondary storage device all data items which have been collected or accumulated at the system level. This data should be retained for a user-determined duration in order to satisfy historical reporting requirements.
2. Backing up of Management Statistical Data. Backup capabilities were described in SPS V, Subsection 3.1.1.

Purpose of Paragraph:

To specify the need for archiving Management Statistical Data.

Requirements for Necessary New Support Software:

Subsection 5.4.4 of SPS Volume VI briefly describes the MDPRINT sub-program which was proposed to satisfy this paragraph's specifications. Further details of this routine are contained in Section 3.6.3 of this Chapter.

Existing Tools Which Satisfy this Requirement:

All of the systems studied provide a means for back-up of data, as was shown in the discussion and examples pertaining to SPS V, 3.1.1.c. However, it is suggested that a coherent programming system for all Management Data Collection and Reporting functions would be preferable. Since so many of the functions cannot be implemented with existing tools, to use such tools in the few cases where they do exist would require the remaining functions to be fulfilled by a multitude of special purpose programs. Instead, a single programming system would eliminate duplication and offer standardization of approach. Such a program could be interfaced as necessary with individual operating systems.

3.6.2.e Full Reporting

Classification: Full

Requirement:

Full Reporting - Provide a report generation facility to output information in a convenient and meaningful manner. This facility must provide the following capabilities:

1. Producing Program Module Statistics reports for the various hierarchical levels including file, program, subsystem and system. These reports will contain such information as net lines of source code, total program source files, total update runs performed in a program and total lines of source code input for a program during a given reporting cycle.
2. Producing detailed and summary Computer Utilization reports. The detailed reports will contain information such as the average computer turnaround time for each programmer. The summary report will contain information such as the average computer turnaround time for all programmers on a project.

3. Producing Program Maintenance Statistics reports for the various hierarchical levels including file, program, subsystem and system. These reports will contain information such as specification omission errors, implementation misinterpretation errors, and number of update runs.
4. Producing Program Structure reports in both detailed and summary form. The detailed reports will list the total tree structure of the program starting at the file specified by the user and including all "included" and "called" source files. The detailed reports will also contain a cross reference listing of all included and called files. The summary reports will contain a partial tree structure showing only the top user-specified number of levels. They will also contain a cross-reference listing.
5. Producing Historical reports from the archival projects data. The content of these reports will be user-indicated, and may include such data as project name, date range, and range of project size expressed in number of program source files. The reports will also contain cost data and key historical data items from the Program Module Statistics, Computer Utilization, Program Maintenance, Statistics and Program Structure reports for both active and inactive software projects.
6. Producing Combination reports for the various hierarchical levels including program subsystem and system. These reports will combine cost data with information from more than one report class (e.g., Program Module Statistics and Program Maintenance Statistics).
7. Producing Cyclic versions of the reports described in Items 1 through 6 above. These reports will include data such as total lines of source code for a program for the given program cycle. They will also flag discrepancies, e.g., when travel costs expended to date exceed the travel costs budgeted to date.
8. Allowing the user to add output routines to the PSL. These routines will be used to output the actual data collected by the user routines which were added in order to satisfy unique user requirements.
9. Producing reports on an exception basis, without user request, when significant variances exist between Actual and Plan data.

Purpose of Paragraph:

To specify the content of the various reports that should be produced by the Management Data Report function of the PSL.

Examples:

Examples of the type of report desired may be found in SPS Volume IX, pages 3-17 through 3-31, while a detailed description of the report contents is contained in Appendix B of the same volume. It should be noted that the format of the reports is not always specified. Histograms and/or line graphs are suggested as an excellent format, particularly for comparison of Plan and Actual data.

It is further suggested that the reports include physical storage requirements for all data and associated software, both in terms of Plan estimates and Actual usage. This would allow management to better understand utilization of peripheral storage devices.

Program Requirements for Necessary New Software:

A summary of software requirements for report production is given in subsections 5.4.3 and 5.4.4 of SPS Volume VI. Further details are described in Section 3.6.3 of this Chapter.

3.6.3 Additional Suggested Specifications for Management Data Collection and Reporting Programs and Data Structures

Although SPS Volumes VI and IX contain specifications on the Management Statistical Data Base (MSDB) and the software required to manipulate it, those specifications are incomplete. This section offers a more detailed examination of the data structures and manipulative programs needed by the MSDB. It is intended as a guideline for construction of the programming system which supports Management Data Collection and Reporting (MDCR). The specifications contained herein are by no means complete, but should aid in initial system design. It should be noted that they are oriented toward on-line operation, since that is an environment more suited to PSL management than is batch. Furthermore, batch implementation is already detailed in the SPS.

It is assumed that the MDCR System will itself be written by a Government contractor, and will be part of the Full PSL Package supplied to prospective contractors by the Government.

3.6.3.1 Data Structures

The Management Statistical Data Base (MSDB) consists of Unit Accounting Records (UAR), the Management Data File, and two report tables. Each UAR is part of what the SPS calls a "Library File", a file which contains information on each active project subdivision (reference SPS VI, Figure 5-4). Each "unit" (i.e., segment of source code stored as a file somewhere in the PSL) within a subdivision has its own accounting record, the general format of which is shown in Table 5-3 of SPS VI. Modifications of that format are suggested in Figure 3.6.3-1, as Table 5-3 omits "unit ending date" but includes "key use" as data items. The former data item is required in several unit-level reports, whereas the latter is not required in any of the outlined programs.

ITEM NUMBER	DATA ITEM	ITEM LENGTH IN BYTES	DATA TYPE	DATA SOURCE	
				BATCH	ON-LINE
1	UNIT NAME	40	CHAR.	CONT.CARD	INPUT PARAM.
2	VERSION	1	INTEGER	A	A
3	MODIFICATION	1	INTEGER	A	A
4	PROGRAMMER	40	CHAR.	JOB CARD	USER RESPONSE
5	UNIT START DATA	6	CHAR.	A	A
6	UNIT ENDING DATE	6	CHAR.	CONT.CARD	USER RESPONSE
7	DATE & TIME LAST MODIFIED	12	CHAR.	A	A
8	LAST PROGRAMMER TO MODIFY	40	CHAR.	JOB CARD	USER RESPONSE
9	SOURCE LANGUAGE	9	CHAR.	CONT.CARD	USER RESPONSE
10	UNIT KEY	6	CHAR.	CONT.CARD	USER RESPONSE
11	RESEQUENCING OP- TION (1 or 0)	1	INTEGER	CONT.CARD	USER RESPONSE
12	TYPE (MAIN, REAL, STUB) (1,2, or 3)	1	INTEGER	CONT.CARD/ A	USER RESPONSE
13	NUMBER OF TIMES INCLUDED BY ANOTHER FILE	1	INTEGER	A	A
* 14	SP EXCEPTION FLAG (1 or 0)	1	INTEGER	A	A
15	SOURCE LINES OF CODE. CODE FROM ANOTHER SOURCE	2	INTEGER	A	A
16	LOC ADDED	2	INTEGER	A	A
17	LOC CHANGED	2	INTEGER	A	A
18	LOC DELETED	2	INTEGER	A	A
19	LOC INPUT (TOTAL)	2	INTEGER	A	A
20	LOC INPUT (REP. CYCLE)	2	INTEGER	A	A
21	# UPDATE EXECU- TIONS (TOTAL)	2	INTEGER	A	A
22	# UPDATE EXECU- TIONS (REP.CYCLE)	2	INTEGER	A	A
* 23	#COMPILATIONS/AS- SEMBLIES	2	INTEGER	A	A
24	START OF USER AREA				
.					
.					
n					

Figure 3.6.3-1. UAR CONTENT

The UAR is serviced by two programs, UPDATE and COMPILE, the general specifications of which may be found in SPS VI, Subsections 5.2.1 and 5.5.1, respectively. More detailed analyses of these programs are given later in this section.

The MDF contains management statistics for programming system levels above that of individual files. Detailed definitions of MDF contents appear in SPS IX, Appendix A. Analysis of that material together with study of SPS VI, Figure 5-10 and Tables 5-4 through 5-7, will yield a moderately good understanding of the structure of this file. It is built to facilitate reporting of programming system development at the system, subsystem and program levels. Note that these reporting levels do not correspond to the hierarchy of project, subdivision and file type levels; rather they represent a conceptual hierarchy for a programming system.

The MDF also provides space for storage of data concerning job runs and for pointers to report tables for cyclic and exception reporting. This latter feature was part of the MDF in the initial SPS design, but insufficient data was included to allow report generation. Since the data format required was different from the rest of the MDF, it seemed most efficient to make the Report Tables separate data structures. They serve as repositories for the data needed to automatically generate reports when certain criteria are met. These criteria may concern time, as in the cyclic reports, or time and value of specified key parameters, as in the exception reports. Outlines for these data structures may be found in Figures 3.6.3-3 and 3.6.3-4. A similar outline for the slightly revised MDF is presented in Figure 3.6.3-2.

The programs which manipulate the MDF, the Cyclic Report Table, and the Exception Report Table are MDFILE, MDUPDATE, MDREPORT, MDAUTORPT, MDPRINT, MDHIST, and MDXCHECK. All of these except MDAUTORPT are generally described in SPS VI, Subsections 5.4 and 6.4. One program contained in those specifications, MDUSER, was found to be redundant and so was omitted.

General functional descriptions of programs affecting the MSDB are given in Figure 3.6.3-5. More detailed analyses are presented in the remaining pages of this section.

##MGMTDATA		000000
CYCLIC RPT. TABLE LOC.		Pointer to CR Table
EXCEPTION RPT. TABLE LOC.		Pointer to ER Table
Level	Record name	Pointer to data record
.	.	.
.	.	.
.	.	.
System level data record **		
Item id*	Item value	
.	.	
.	.	
.	.	
Job level data record		
Job #	Item id*	Item value
Subsystem level data record format same as system level.		
Program level data records		
Program id	Item id*	Item value
.	.	.
.	.	.
zzzzzz##		Unused

Figure 3.6.3-2. Management Data File Structure

- * Item id would probably be arbitrary number and a letter to indicate if Actual or Plan data.
- ** Input items from MDREPORT and MDXCHECK not contained in data record.

CYC. RPT. TABLE

Project Name

Report Mnemonic	Start Date	Cycle Type	Date of Next Rpt.	Record Name
.
.
.
.
ENDCRT ##			Unused	

Figure 3.6.3-3. Cyclic Report Table.

EXC. RPT. TABLE

Project Name

Report Mnemonic	Start Date	Cycle Type	Date of Next Rpt.	Record Name	Items	
					Ident.	Variance
					.	.
					.	.
					.	.
.
.
.
ENDERT ##			Unused			

Figure 3.6.3-4. Exception Report Table

Program Name	Functional Description
ALLOCATE	Allocates space for new Library Files (i.e., file types) Sets up Library File entries Indicates if management data to be collected
COMPILE	Prepares input to precompiler Invokes precompiler and compiler Updates UAR(s) as needed
* MDAUTORPT	Checks report tables Generates list of automatic reports, both cyclic and exception, to be printed
* MDFILE	Allocates and initiates MDF Terminates automatic data collection (ADC) Initiates ADC on project and/or subdivision level
* MDHIST	Produces a Historical report on basis of archival system level reports
* MDPRINT	Prints all reports except for Historical
* MDREPORT	Sets up Cyclic Report Table
* MDUPDATE	Allows alteration of MDF data and of UAR user area data
* MDXCHECK	Sets up Exception Report Table
* UPDATE	Generates and maintains PSL data files (units) Initiates and updates UAR

Figure 3.6.3-5. Table of programs affecting MSDB.
Starred items further described in this report.

3.6.3.2 Software Specifications

The following is a description of those programs which were starred in Figure 3.6.3-5. Presentation of each program will include:

- 1) discussion of the functions to be performed and various software necessities,
- 2) input,
- 3) output,
- 4) flow chart of program.

The program affecting the UAR is discussed first, followed by those programs which concern the MDF. Order of presentation in this group is the same as SPS VI, Section 5.4

Once again it should be emphasized that these are intended only as guidelines and are not complete.

3.6.3.2.1 Update

UPDATE is responsible for initializing and updating program units, i.e., files. It also updates the UAR when changes are made to a file. This differs from the original SPS specifications. The alteration was made because it is more efficient particularly in an interactive environment, to update information pertaining to a file at the time a file itself is updated. UPDATE consists of five main functions: Adding a new file to the PSL, purging a file, replacing a file, editing a file, and copying.

- 1) Addition function:

This command allows a new file to be added to the PSL and its UAR to be set up if the MDC option is YES. Command format is

UPDATE ADD filename

where "filename" is the name of the file, i.e., unit, to be added.

- 2) Purge function:

This command purges a file and its UAR, if any, from the PSL. Command format is

UPDATE PURGE filename

where "filename" is the name of the file to be purged.

3) Replace function:

This command allows data in an existing file to be replaced by data in another file or by data specified on-line. The command format is

UPDATE REPLACE filename

where "filename" is the name of the file which is to have its data replaced.

4) Edit function:

This command allows a file to be edited and the file to be altered by adding, deleting, or modifying records. It also updates the UAR accordingly. Command format is

UPDATE EDIT filename

where "filename" is the name of the file to be edited.

5) Copy function:

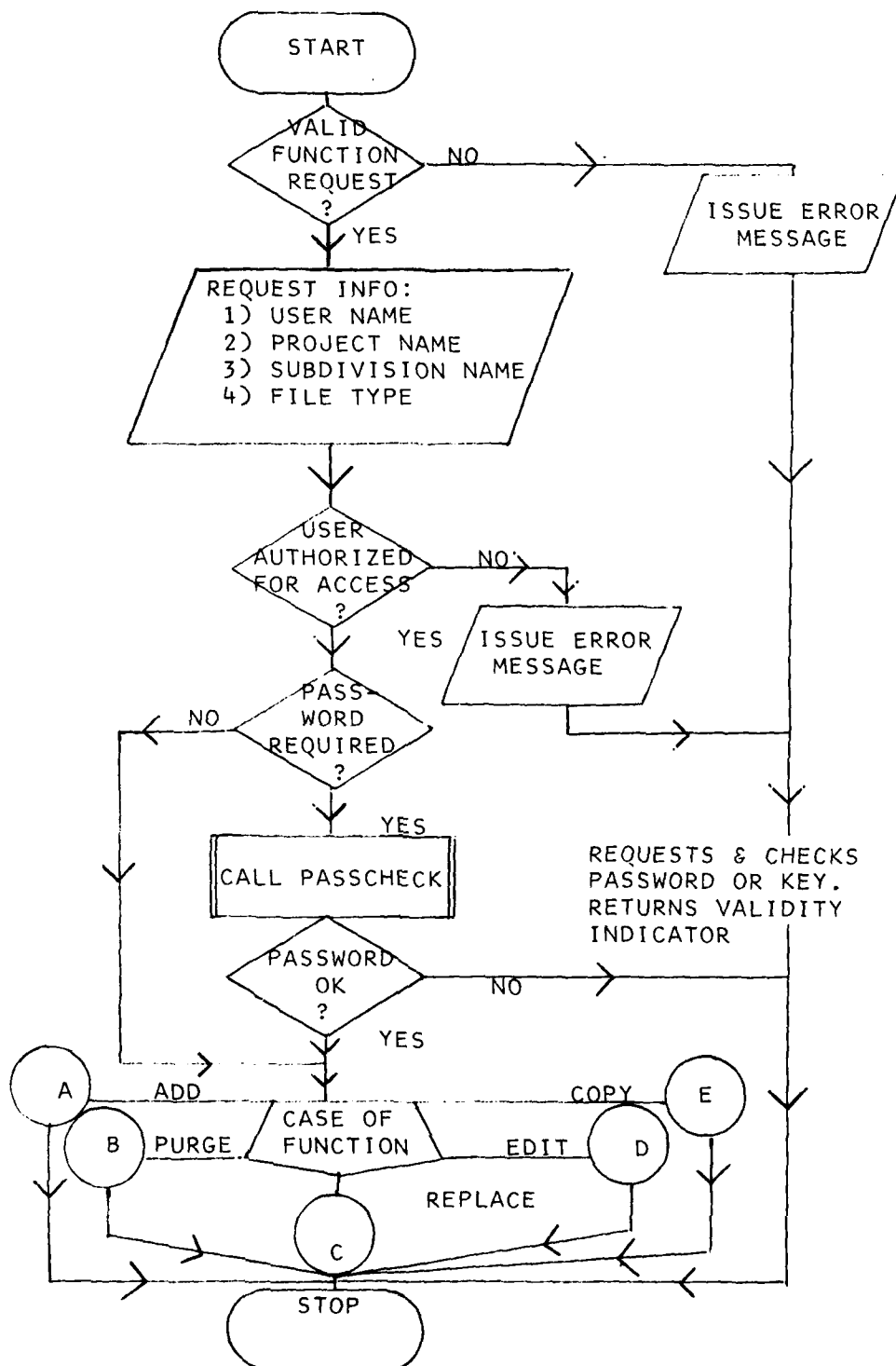
This command causes the formation of a copy of an already existing file. The name of the copy is user designated via interactive dialogue. The UAR for the copy is also generated. Command format is

UPDATE COPY filename

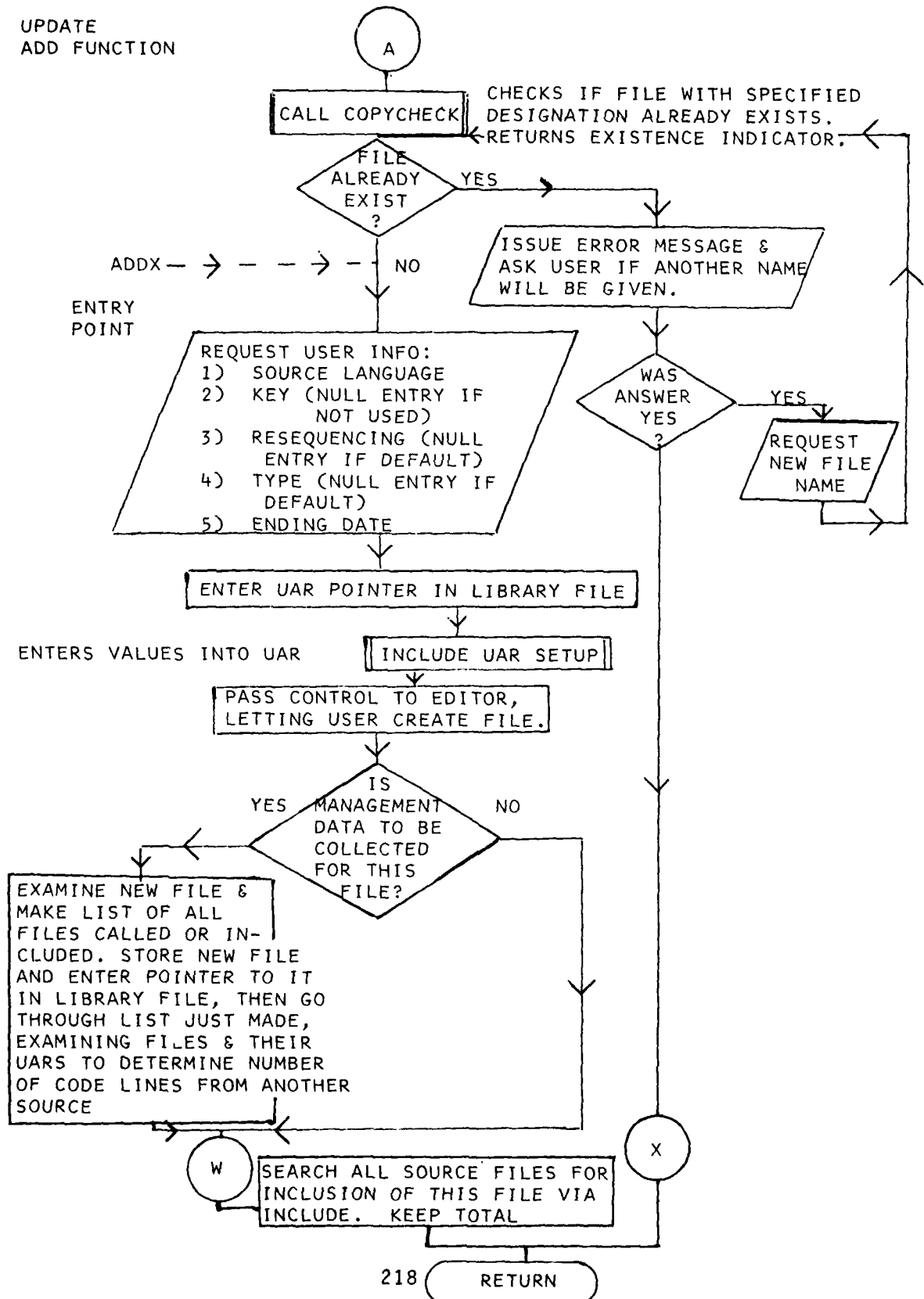
where "filename" is the name of the file to be copied. If filename is "ALL" then all files of user-specified type are copied, retaining the same file names.

Input: function, file name
interactive responses:
password (if requested)
ADD: source language
key (optional)
resequencing indicator
unit type (MAIN, STUB, etc.)
ending date
etc.
PURGE: key (if requested)
REPLACE: key (if requested)
new data or location thereof
EDIT: key (if requested)
COPY: new file type, subdivision, and project
new filename (if input parameter = ALL)

UPDATE



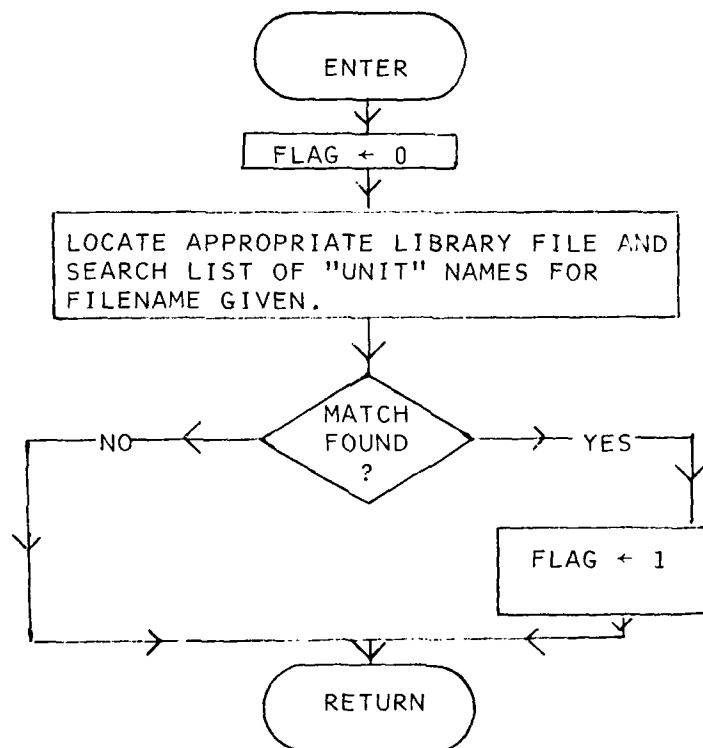
UPDATE
ADD FUNCTION



UPDATE SUBR.:
COPYCHECK

INPUT PARAMETERS: FILENAME, TYPE, SUBDIVISION, PROJECT

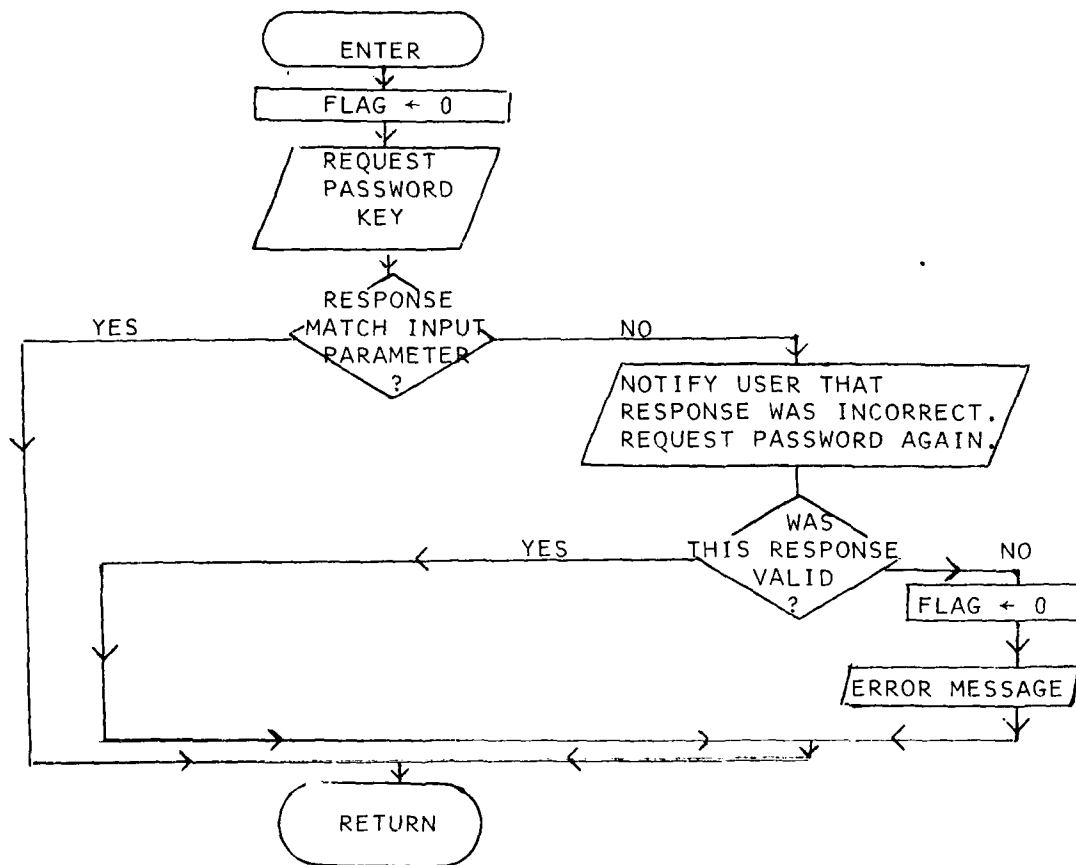
OUTPUT PARAMETERS: FLAG INDICATING IF FILE ALREADY EXISTS



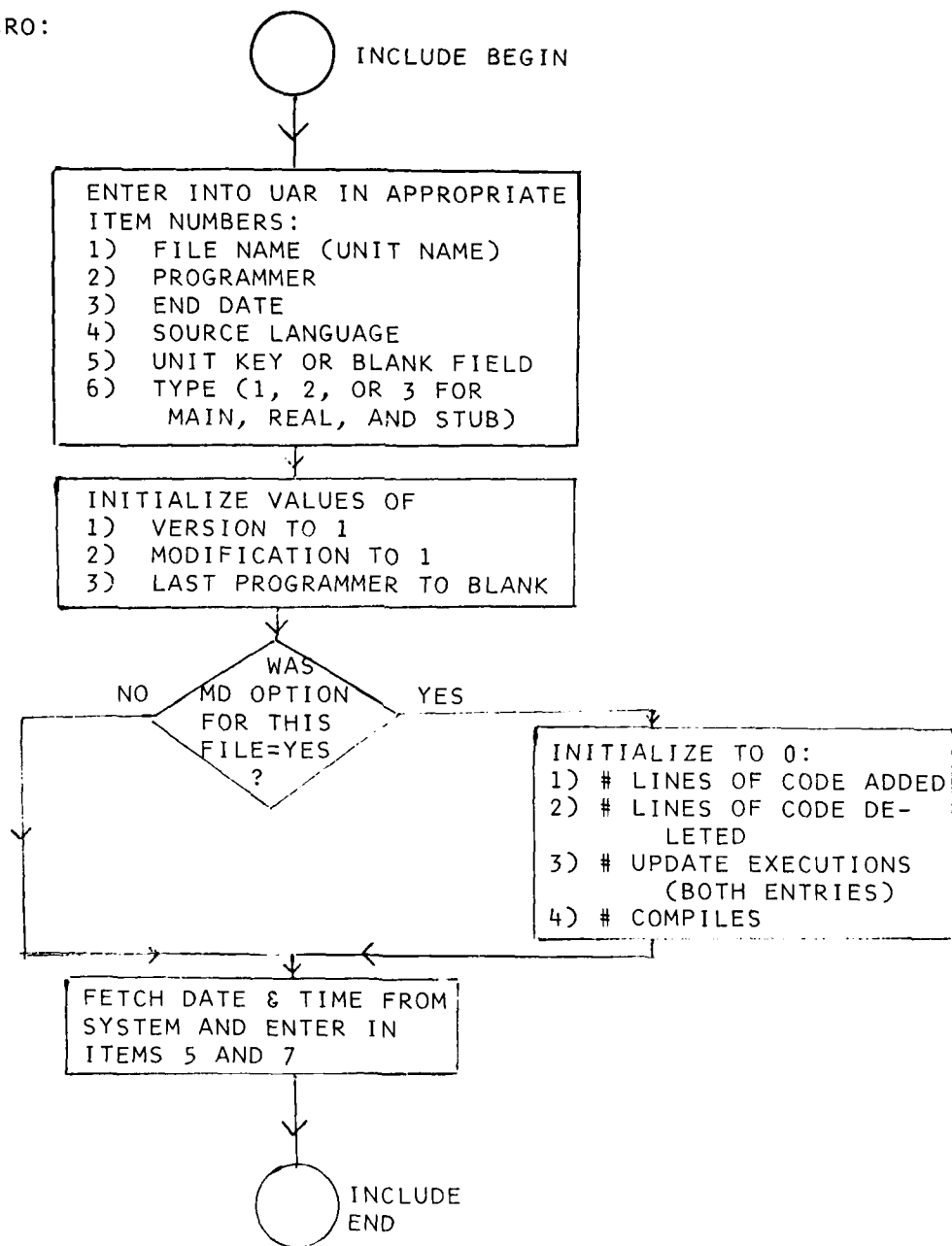
UPDATE SUBR.:
PASSCHECK

INPUT PARAMETERS: PASSWORD KEY FOR FILE IN QUESTION

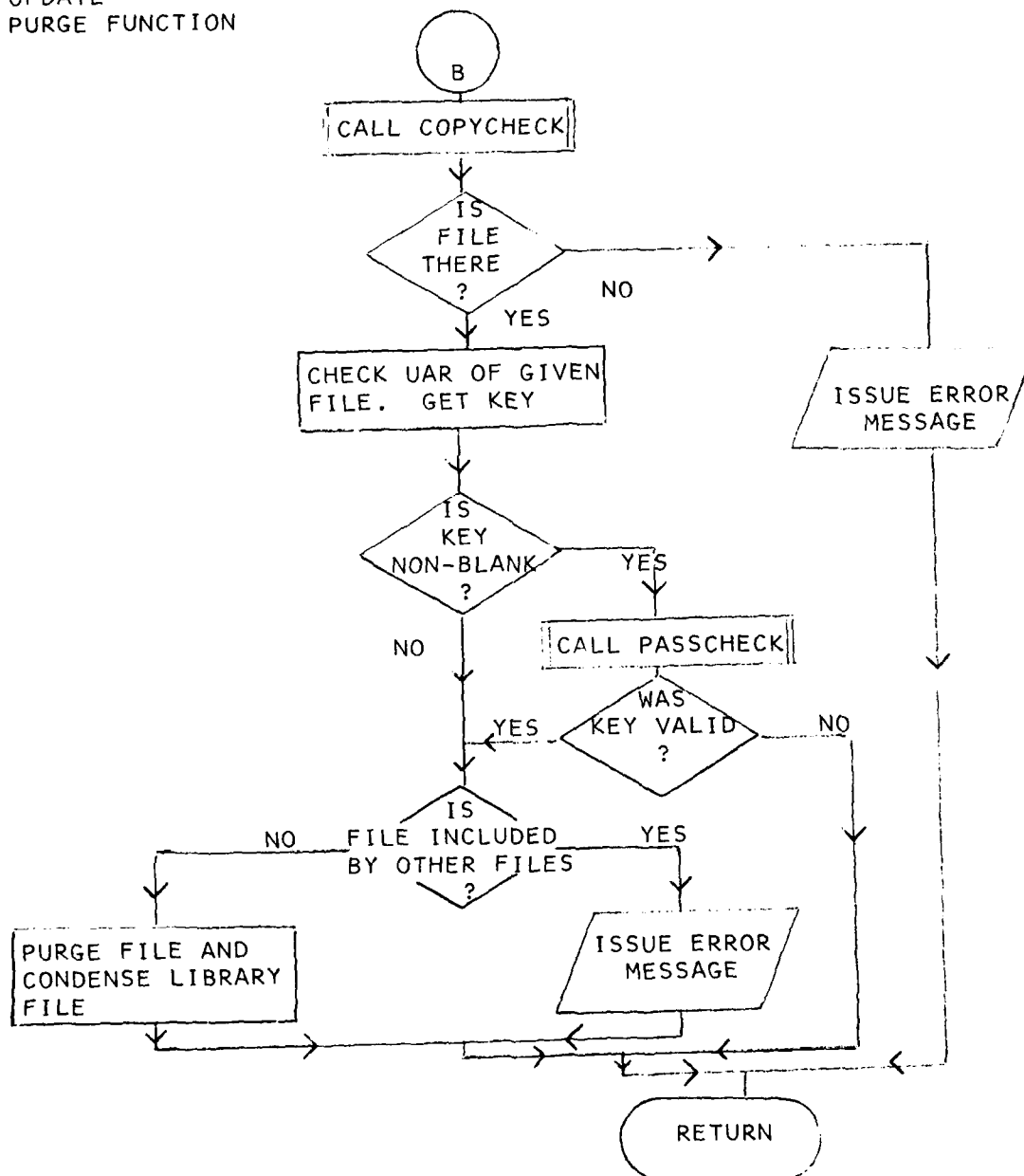
OUTPUT PARAMETERS: FLAG INDICATING VALIDITY OF PASSWORD KEY



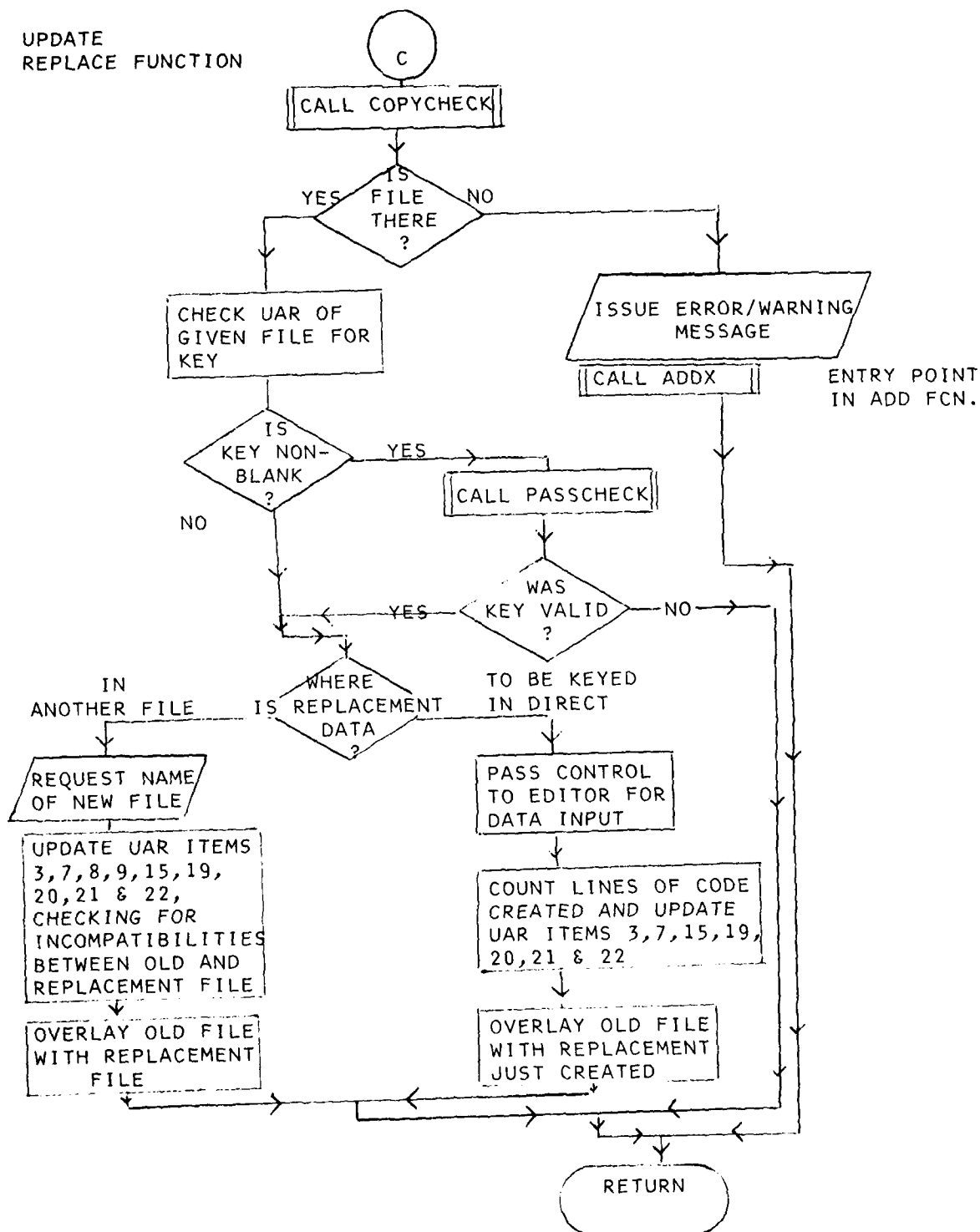
UPDATE MACRO:
UAR SETUP



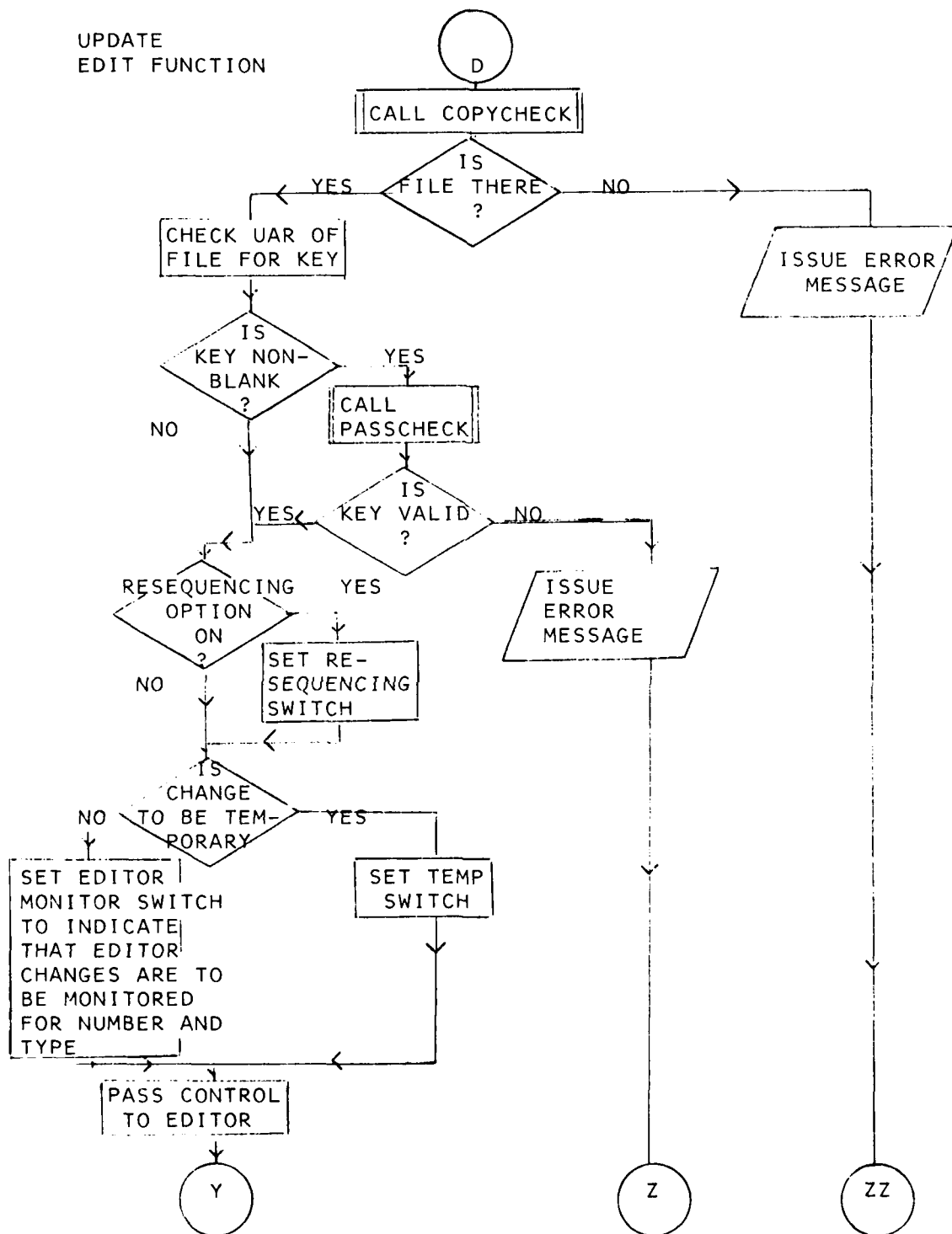
UPDATE
PURGE FUNCTION

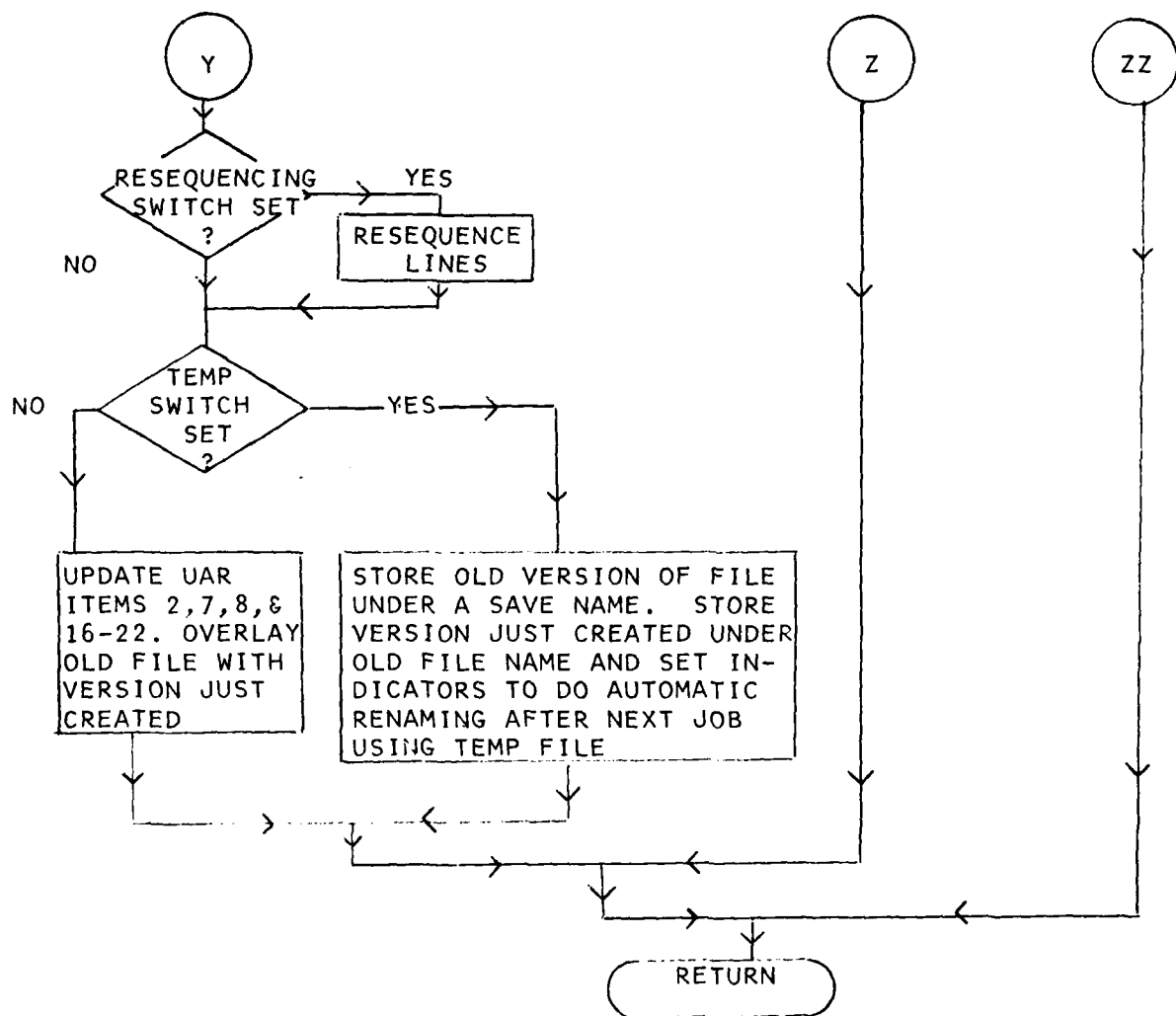


UPDATE
REPLACE FUNCTION

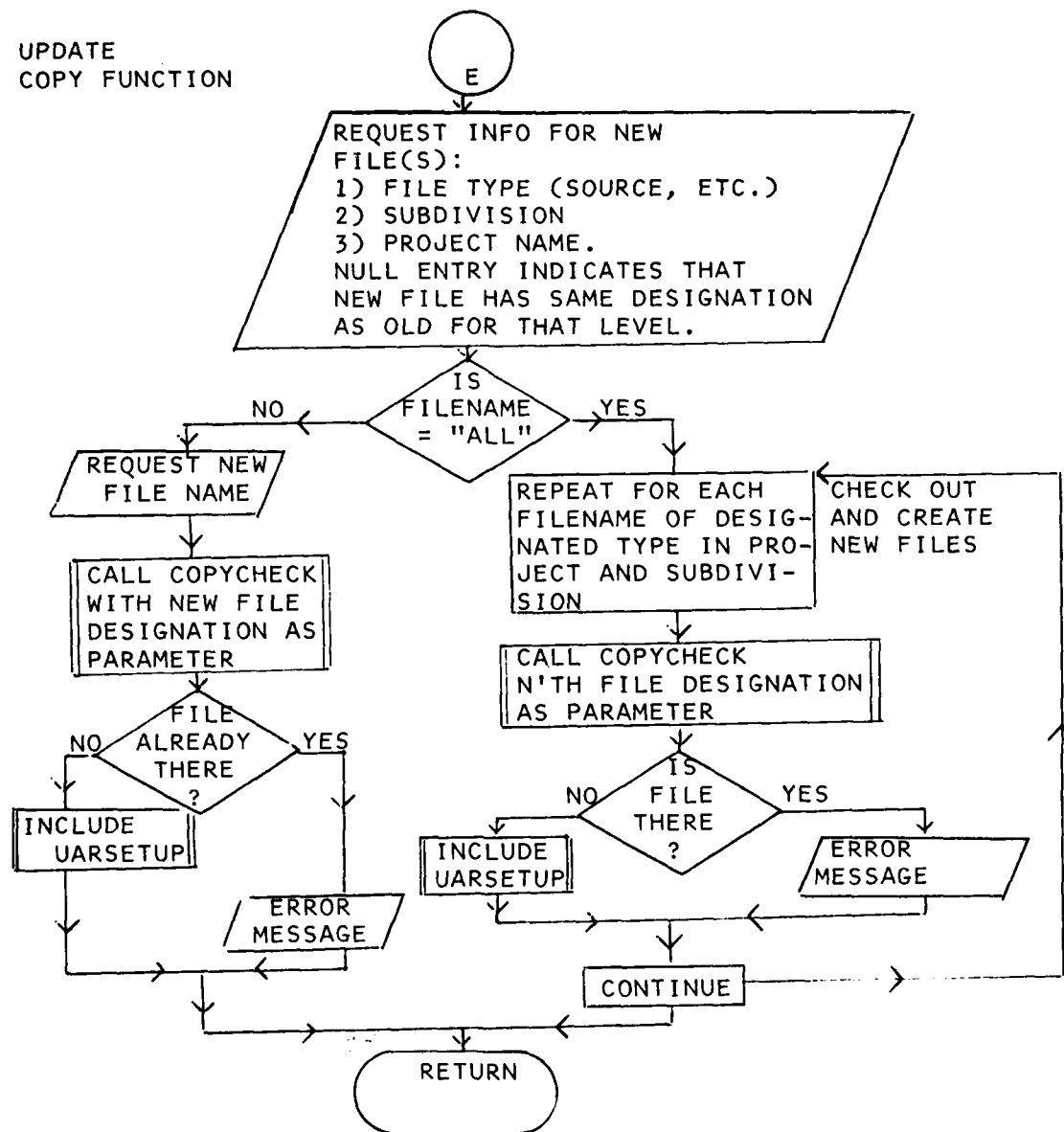


UPDATE
EDIT FUNCTION





UPDATE
COPY FUNCTION



3.6.3.2.2 MDFILE

MDFILE initiates the Management Data File (MDF) for each project, checks that named files in a project have the Management Data option set, reallocates storage space for the MDF when necessary, and terminates collection of management data.

1) Allocation function:

This command allocates storage space for the MDF, and is used when setting up the data collection function for a new project. Command format is

```
MDFILE ALLOCATE project name volno. [# entries]
```

where:

project name is the name of the project for which management data is being collected.

volno is the identification of the direct access volume used for storage.

entries is an optional parameter specifying the number of entries in the file. If not given, a default is used.

2) Reallocation function:

This command is used to reallocate MDF storage space if it is insufficient. Command format is:

```
MDFILE REALLOCATE project name volno [# entries]
```

where:

project name is the name of the project for which management data is being collected.

volno is the identification of the direct access volume used for storage.

entries is an optional parameter specifying the number of entries in the file. If not given, a default of approximately 1.25 the preceding number of entries is used.

3) Collection initiation function:

The ALLOCATE program (see Figure 3.6.3-5) generates the File Accounting Record for each type of file within a subdivision. This generation includes setting the Management Data Collection option for that file type. However, ALLOCATE sets the MDC option only for a specified file type within a specified subdivision, not for an entire subdivision or project. If the latter is desired, or if the user wishes to set a previously clear MDC option, then the MDFILE subcommand "MGMTDATA" is used. The command format is

```
MDFILE  MGMTDATA  projectname  subdivision  file type
                                ALL          ALL
```

where:

project name is the name of the project for which management data is being collected.

subdivision is the name of the project subdivision (the SPS terms it "library") which is to have the MDC option set. If ALL is specified, then all subdivisions within a project have the option set.

filetype is the type name (SOURCE, etc.) of the files that are to have the MDC option set. If ALL is specified, then all file types within the subdivision(s) (except for OBJECT and LOAD) have the MDC option set. This subcommand generally functions to check the Management Data Collection option in the File Accounting Record (see SPS VI, Figure 5-4 and Table 5-2), and to set or reset that option to YES for the designated collection levels.

4) Collection termination function

This command terminates Management Data Collection at the specified collection levels by setting the MDC option to NO in the appropriate File Accounting Records. Command format is

```
MDFILE  ENDMGDATA  projectname  {subdivision}  {filetype}
                                ALL          ALL
```

where the parameters are defined as for the MGMTDATA command, except that the MDC option is cleared, i.e., set to NO.

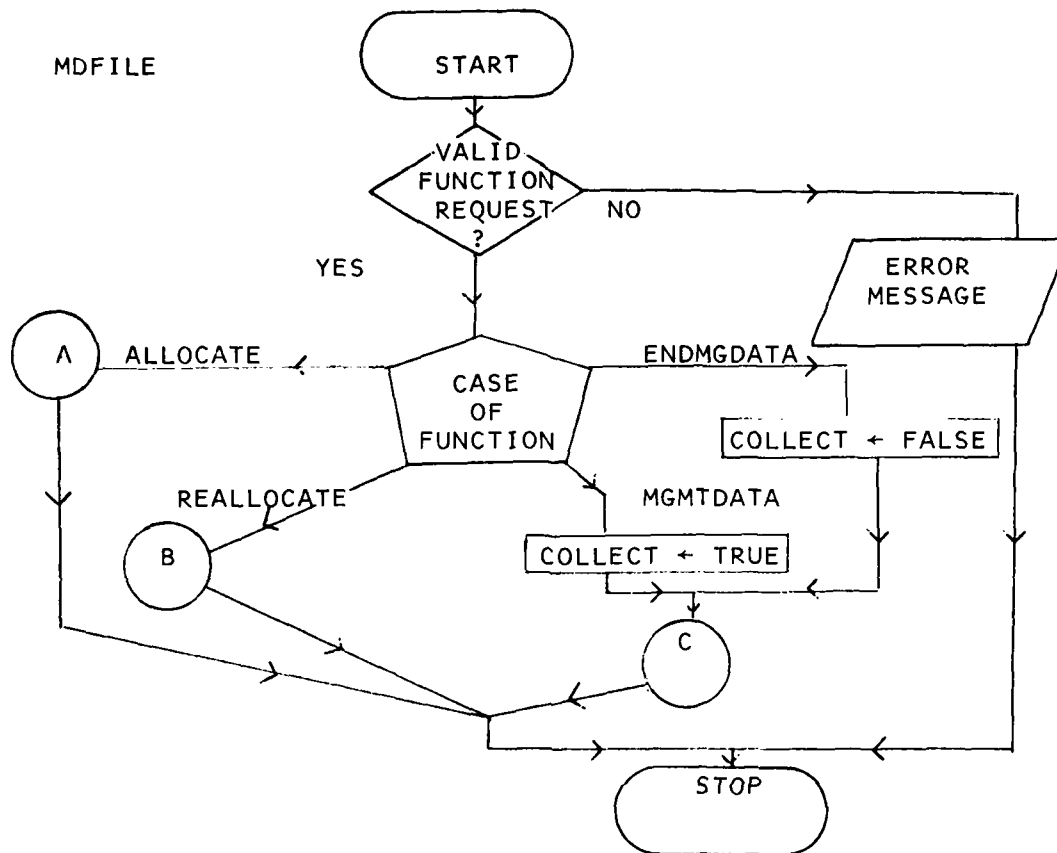
Input:

- 1) function
- 2) project name
- 3) volume # for functions 1) and 2)
subdivision or ALL for functions 3) and 4)
- 4) # entries (optional) for functions 1) and 2)
file type or ALL for functions 3) and 4)

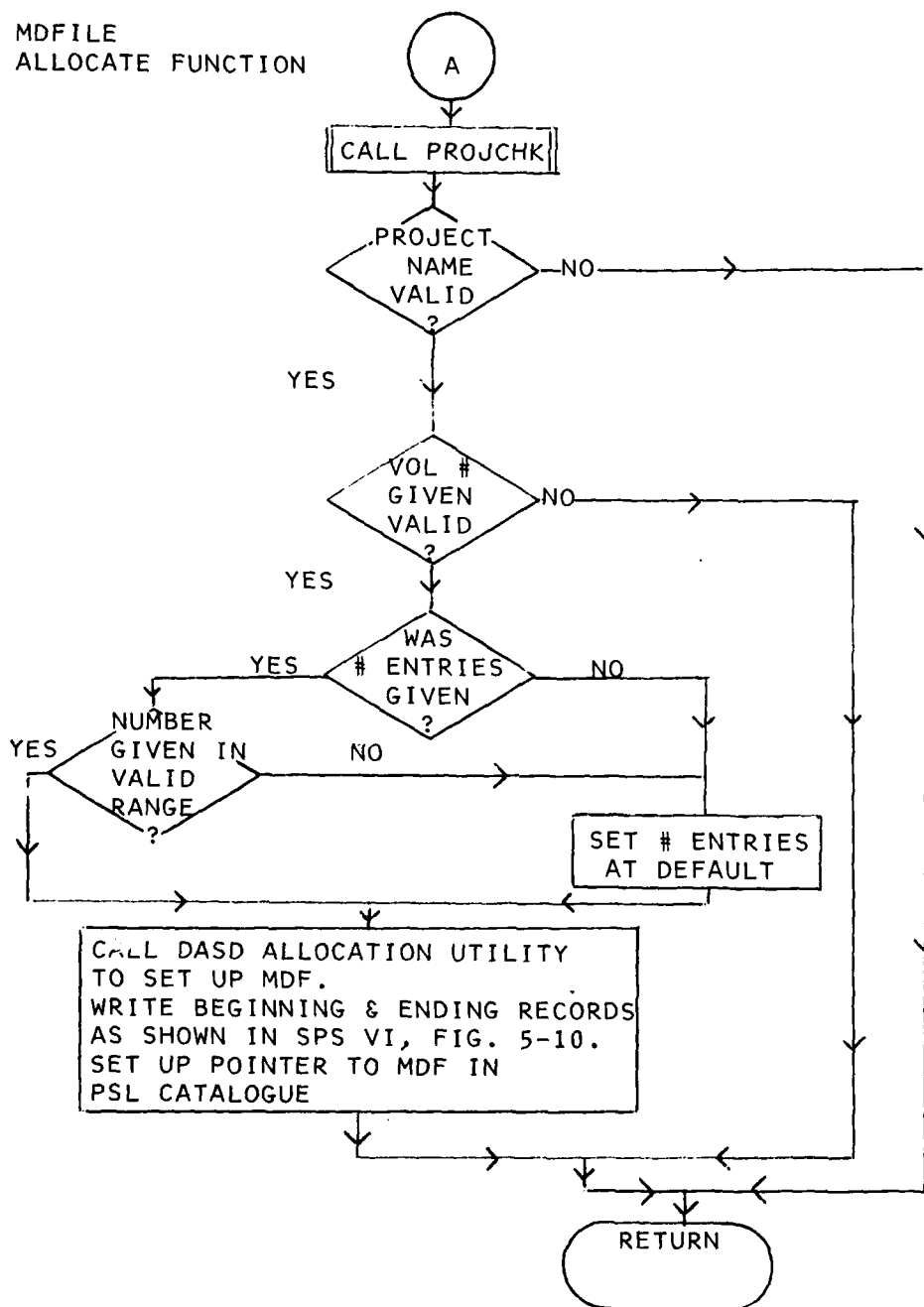
Output:

- 1) storage space for the MDF (functions 1 and 2)
or
- 2) an altered File Accounting Record (FAR) of the
Library File (function 3)
- 3) an altered FAR and MDF deleted (function 4)

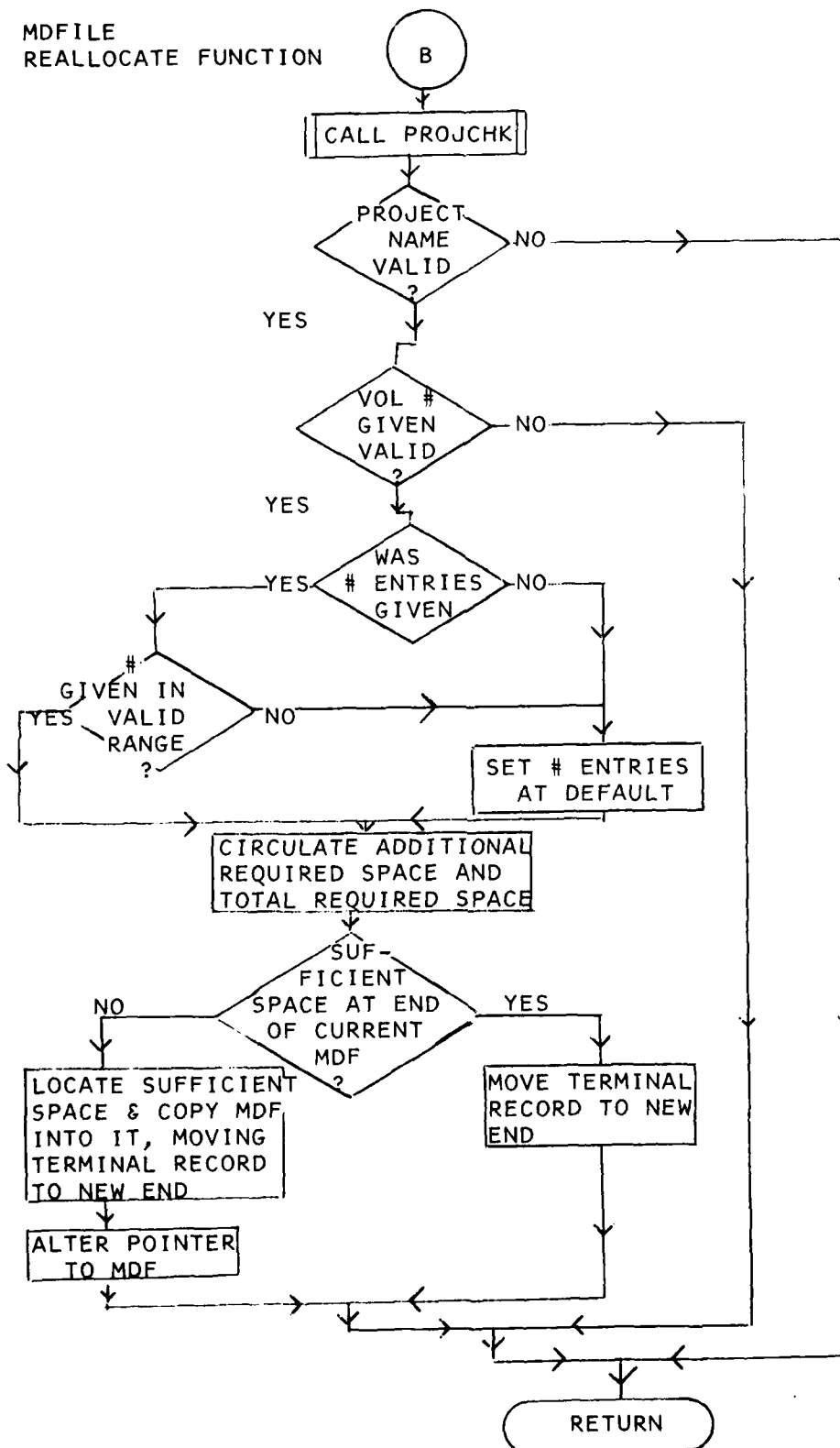
MDFILE



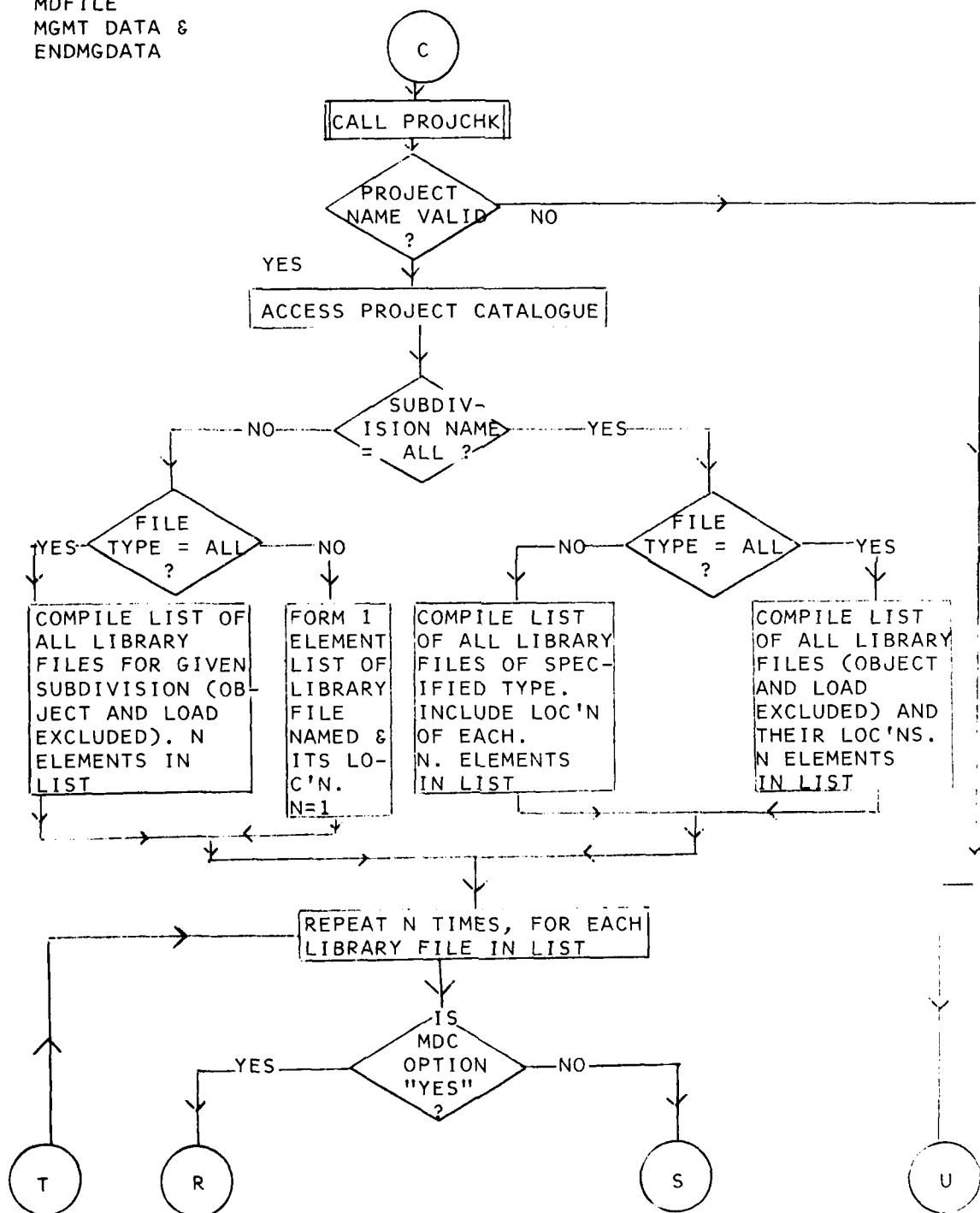
MDFILE
ALLOCATE FUNCTION

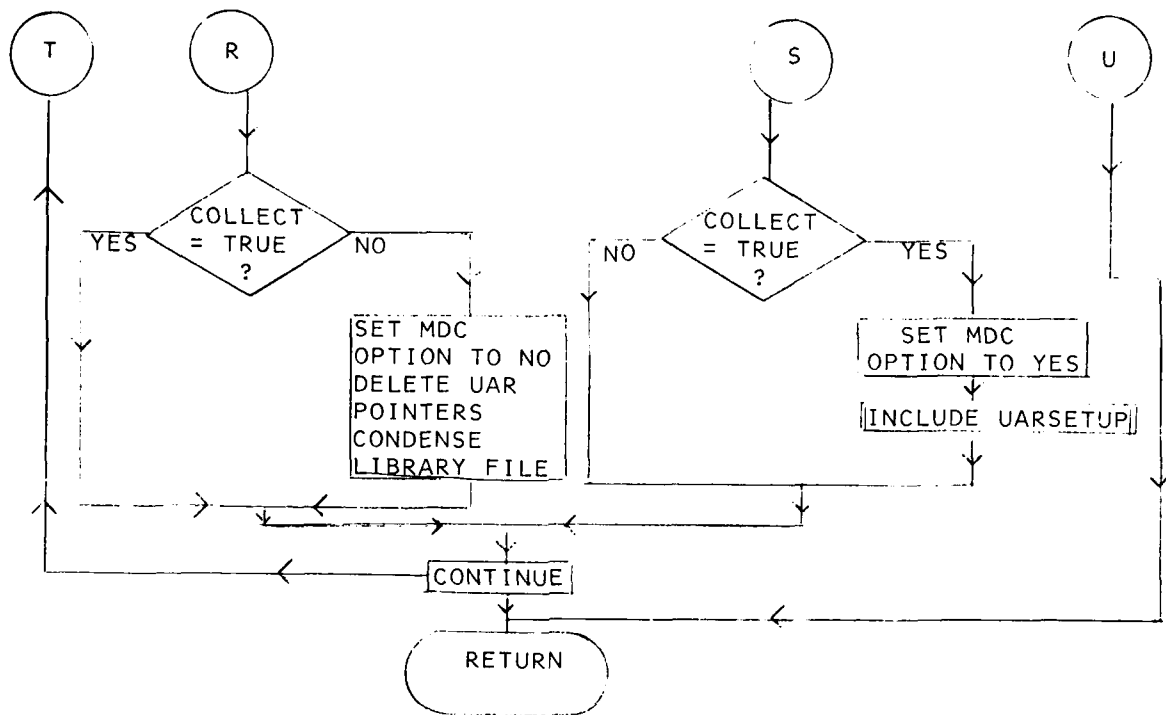


MDFILE
REALLOCATE FUNCTION



MDFILE
MGMT DATA &
ENDMGDATA





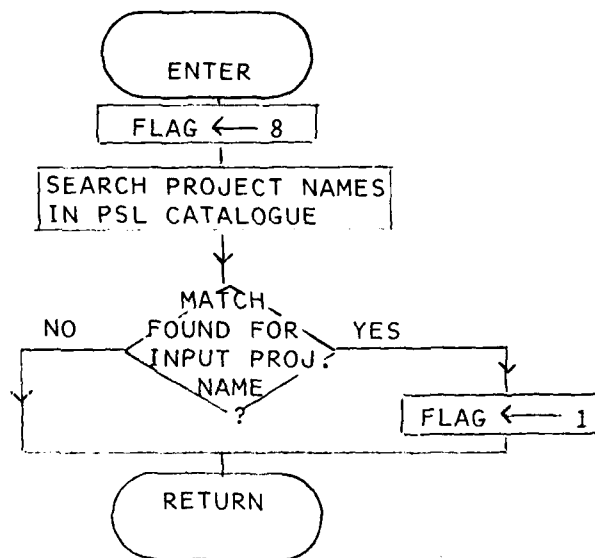
MDPRINT, MDHIST,
MDREPORT, MDXCHECK,
MDUPDAT,
MDFILE SUBR.:
PROJCHK

INPUT PARAMETER:

PROJECT NAME

OUTPUT PARAMETER:

FLAG INDICATING
IF PROJECT HAS
BEEN INITIATED



3.6.3.2.3 MDUPDATE

MDUPDATE allows the user to update (i.e., create, alter, and delete) the MDF, most of which consists of manually input data. It also allows the user to alter the user area of the UAR. Updating may be done on information in any of the five record levels: SYSTEM, SUBSYSTEM, PROGRAM, JOB, and UNIT.

The updating process as herein described assumes a semi-intelligent interactive Editor which can check validity of data item formats. The Editor can be exited in two ways: With and without saving the edited file.

Command format is:

MDUPDATE function projectname record level file name

where:

function is either ADD or DELETE, indicating that the specified record level is to be added/alterd or to be deleted.

project name is the name of the project for which the management data is being allocated.

record level is the level of the management data concerned in this operation. Valid values are SYSTEM, SUBSYSTEM, PROGRAM, JOB, and UNIT.

filename is the name of the file concerned. If JOB level was specified it is the job identifier. If UNIT level was specified it is the filename of the file whose UAR is to be accessed. Otherwise it is the name which is to be assigned to the management data record within the MDF.

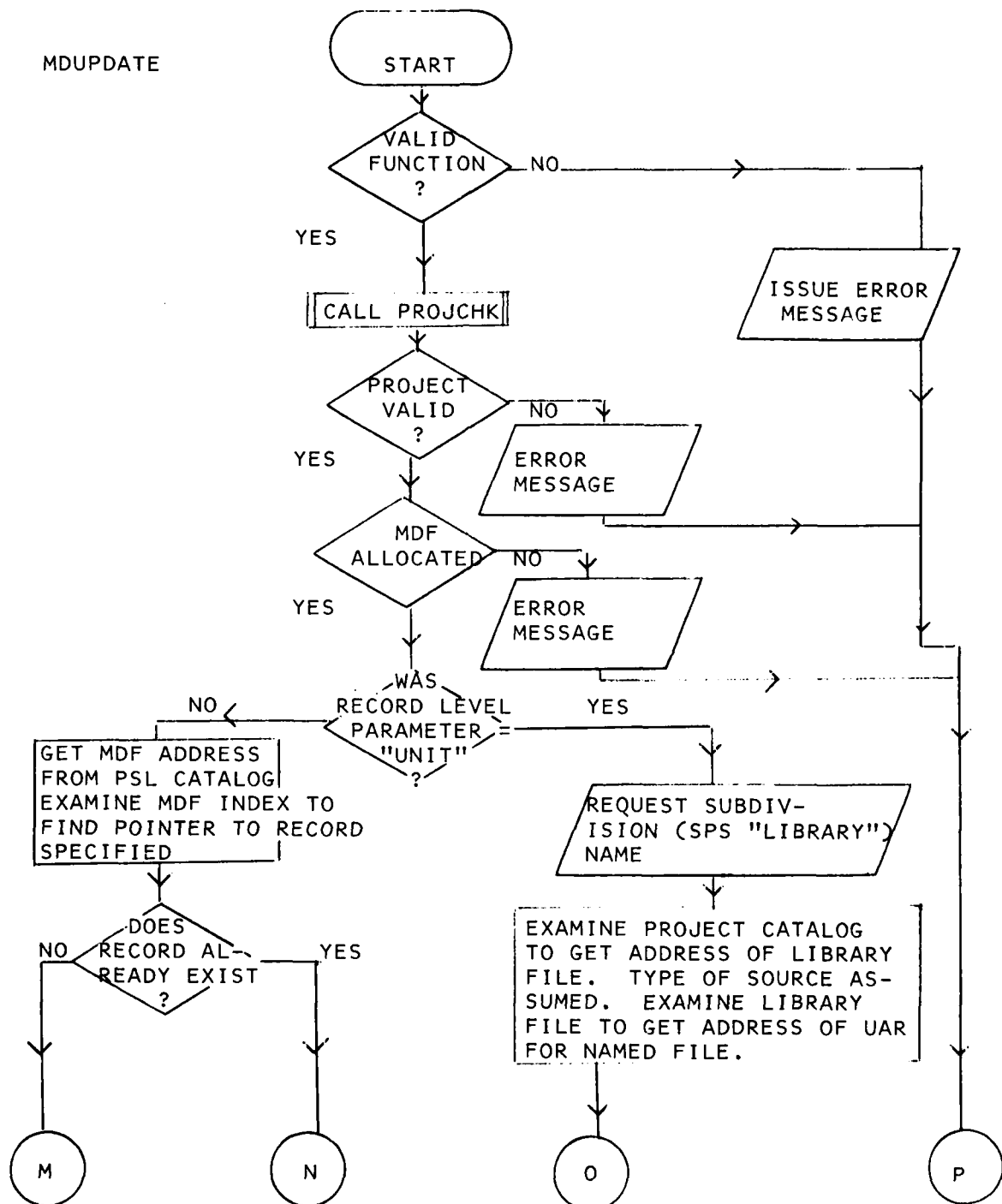
Further information on data contained within the MDF may be found in SPS IX, Appendix A. Initial specifications for MDUPDATE are given in SPS IX, Subsections 5.4.2 and 6.4.2.

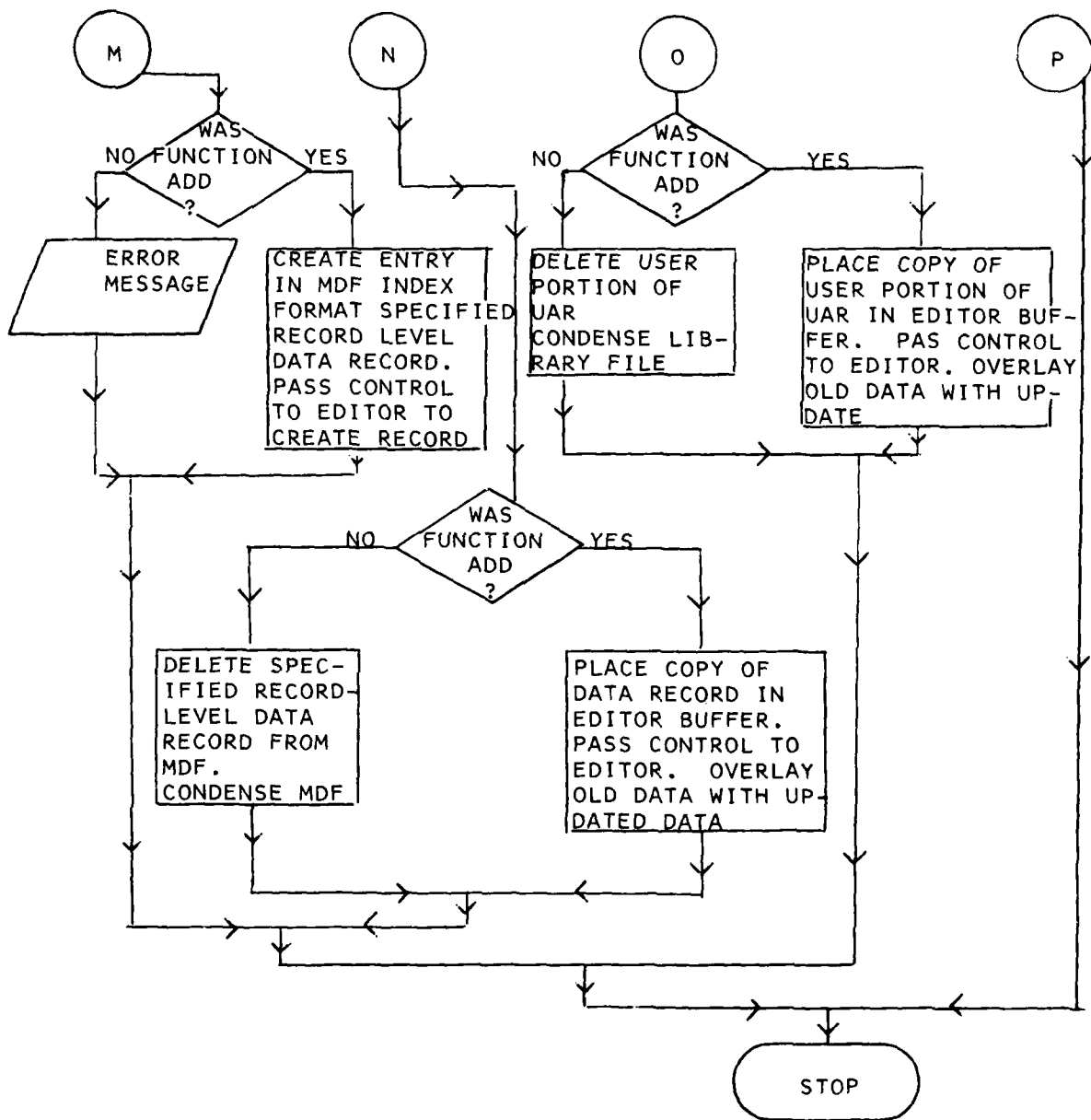
Input:

- 1) updating function
- 2) name of project
- 3) record level
- 4) file name

Output: An updated MDF.

MDUPDATE





3.6.3.2.4 MDREPORT

MDREPORT is responsible for establishing and maintaining the Cyclic Report Table (see Figure 3.6.3-3) for each project. In conjunction with MDAUTORPT it allows automatic generation of specified reports at specified intervals. MDREPORT should be invoked upon initiation of a project for which Management Data is to be automatically collected. Command format is

MDREPORT projectname

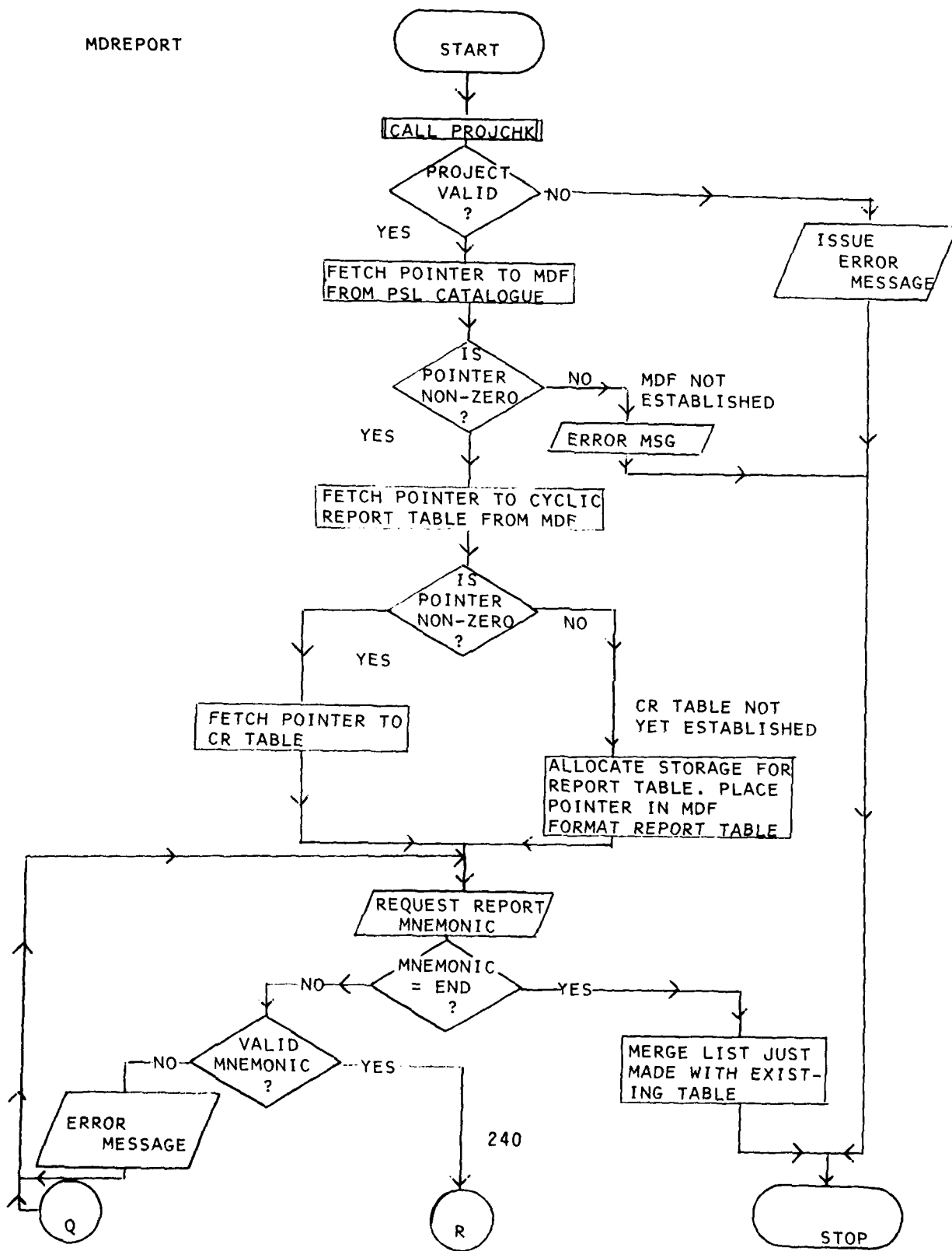
where projectname is the name of the project for which reports are being generated. All other necessary information is requested of the user during interactive program execution.

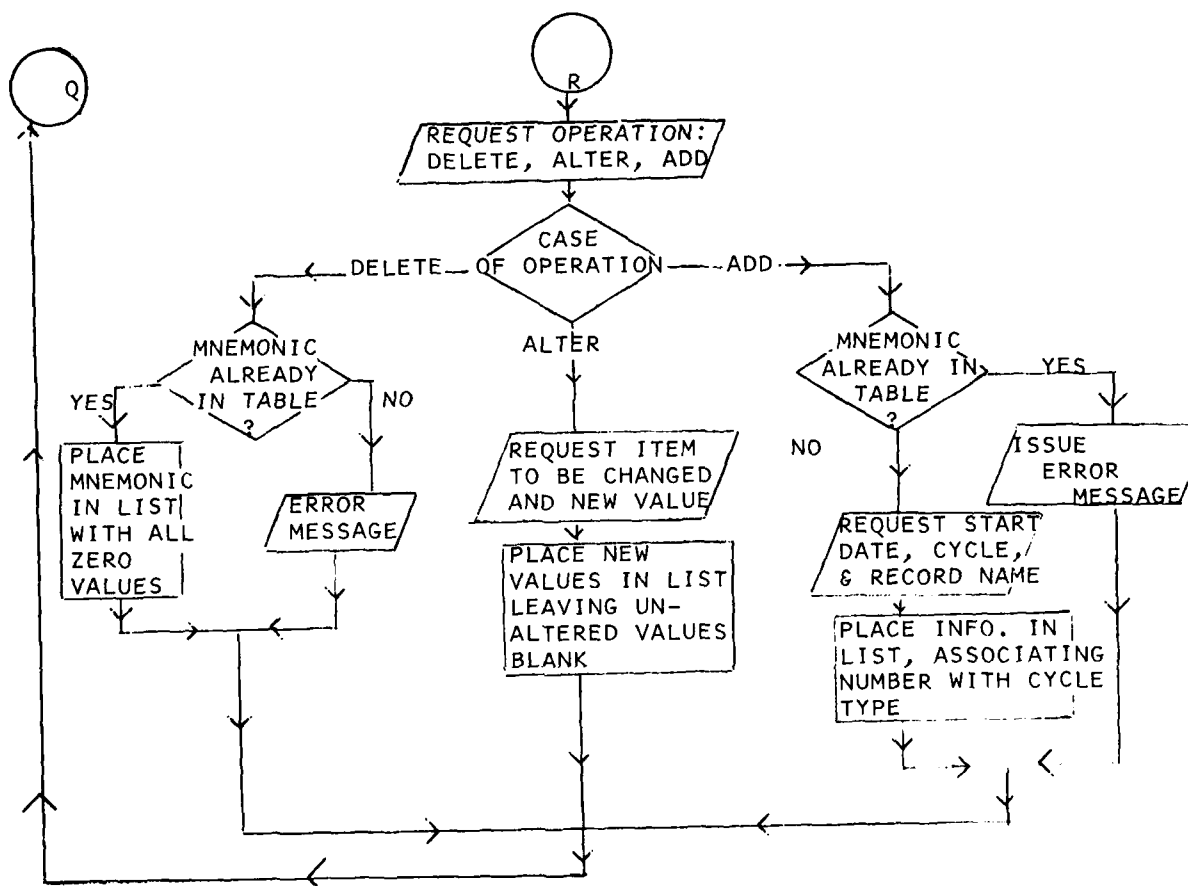
Input: project name

 interactive responses: report mnemonic
 report start date
 report cycle
 record name

Output: Cyclic Report Table, updated.

MDREPORT





3.6.3.2.5 MDAUTORPT

MDAUTORPT scans the report tables, both Cyclic and Exception, to determine if a report is to be automatically generated. If so, it notifies the system operator and requests clearance to print the named reports. This is advisable so that printer facilities may be properly scheduled. When clearance is given, MDAUTORPT updates the appropriate report table and generates the job and control information necessary to schedule MDPRIINT for each report to be automatically printed.

This program does not require user input, and is not initiated by user command. Rather it is an automatically scheduled, daily executed program which requires only operator response for key items. Those responses concern the clearance for printing previously mentioned, and identification of the tape to be mounted for system report generation. This latter may be avoided if a table of archiving tapes is kept internally.

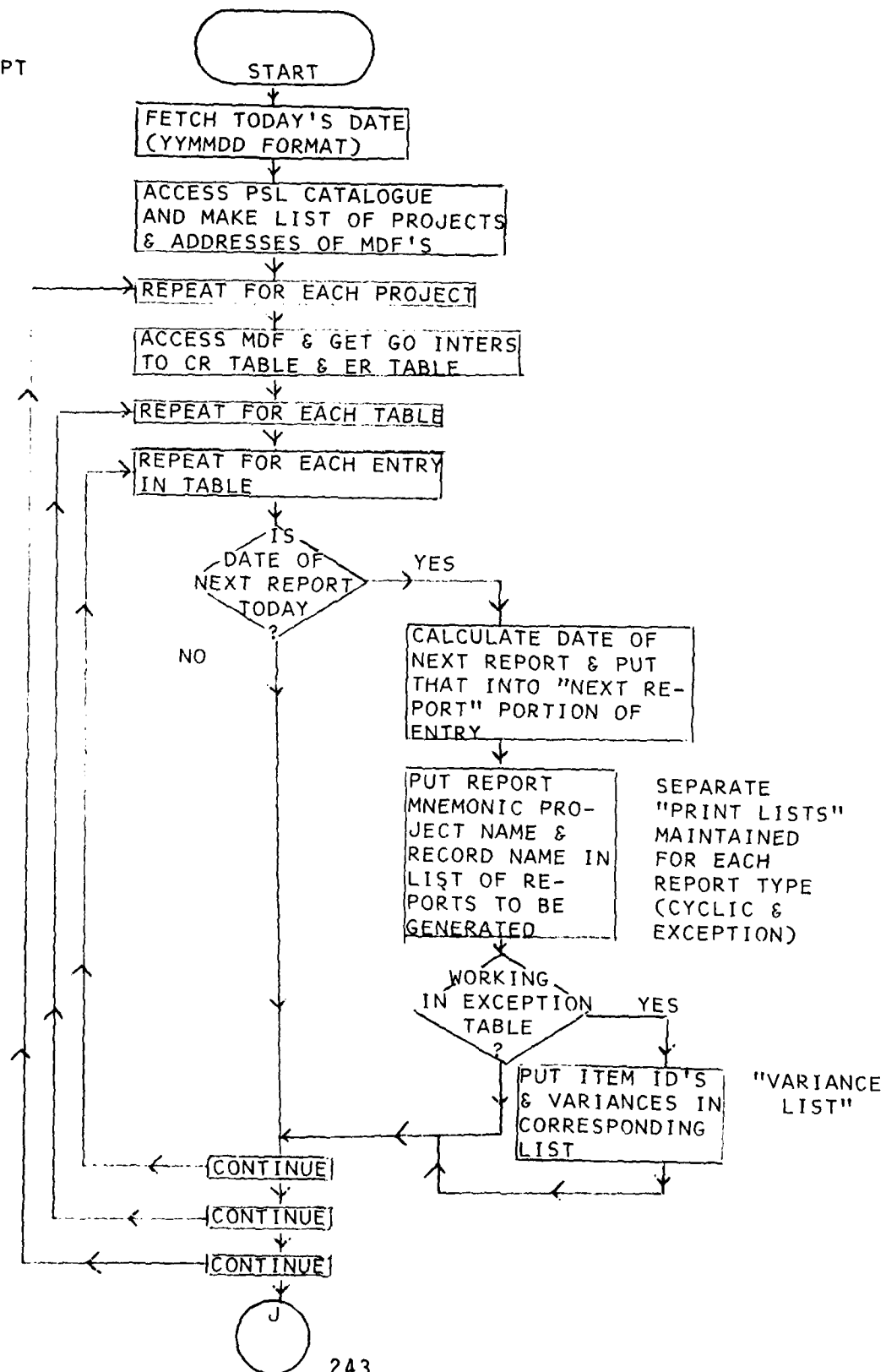
While MDAUTORPT was not included in the SPS specifications, it is obviously needed in order to automatically generate reports, and so is included here.

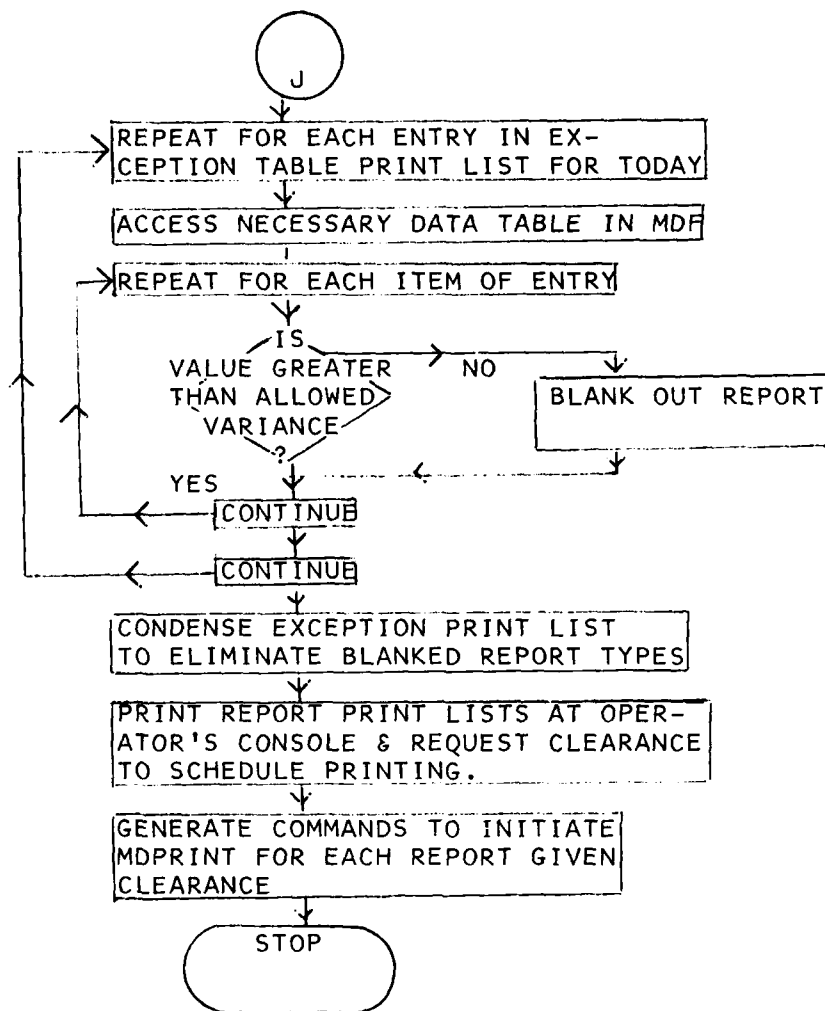
Input: From user: none

From operator: print clearance
tape id for system report, if needed.

Output: Commands to schedule MDPRIINT for each report.

MDAUTCRPT





3.6.3.2.6 MDPrint

MDPRINT is responsible for printing all reports at SYSTEM, SUBSYSTEM, PROGRAM, JOB, and UNIT levels. The full lists of these reports may be found in SPS VI, Table 5.8. Such reports may be printed due to either user request or scheduling by the automatic report generation program, MDAUTORPT. The latter prints reports in both a cyclic and an exception basis (See Section 3.6.3.2.5).

The contents of the various reports is detailed in SPS IX, Appendix B, and selected sample reports illustrated in Figures 3.6-2 through 3.6-7 of the same volume. The report formats shown are strictly tabular, and do not make use of graphical methods of data presentation, e.g., histograms and line graphs. However, an optimally implemented MDCR facility would include report format options that would allow output in graphic form.

The following flow chart shows only the control program. Each report type represented by a different mnemonic would have its own subprogram that takes needed information from the MSDB, calculates any required values, then formats the printout according to the format option selected. The printout should also include a cover page designating the type of report, date, and report initiator (e.g., "Requested by John Jones, Programmer", "Cyclic Report", "Exception Report"). Specifications for these individual report subprograms are not given.

The command format is:

MDPRINT projectname report id word name $\left\{ \begin{array}{l} \text{no. levels} \\ \text{tape no.} \end{array} \right\}$
 $\left[\text{SRC} = \left\{ \begin{array}{l} \text{EX} \\ \text{CY} \end{array} \right\} \right] \left[\text{FO} = \text{format} \right]$

where:

projectname is the name of the project for which the report is being generated.

report-id is the mnemonic of the report being generated (UP, PP, etc.).

recordname is the name of the highest level MDF record on which the report is being made. Multiple values are possible for JOB level reports.

no. levels is an integer specified when a Program Structure report (PS) has been requested. This integer represents the number of levels into the tree structure the report is supposed to cover. If number of levels is not specified, all levels below the named record will be covered.

tape no. is the identifier for the archiving tape utilized in SYSTEM level reports. The mnemonics for these reports are unique in that the first two characters are "SP", "SC", or "SR".

SRC = parameter specifies the source of the report. If not specified, it is assumed to be user requested, and the name and job status of the user (needed for report cover page) is picked up by the system from terminal login information. This parameter will only be specified on commands generated by MDAUTORPT, which will enter the value "EX" if an Exception Report is being printed, and "CY" if a Cyclic Report.

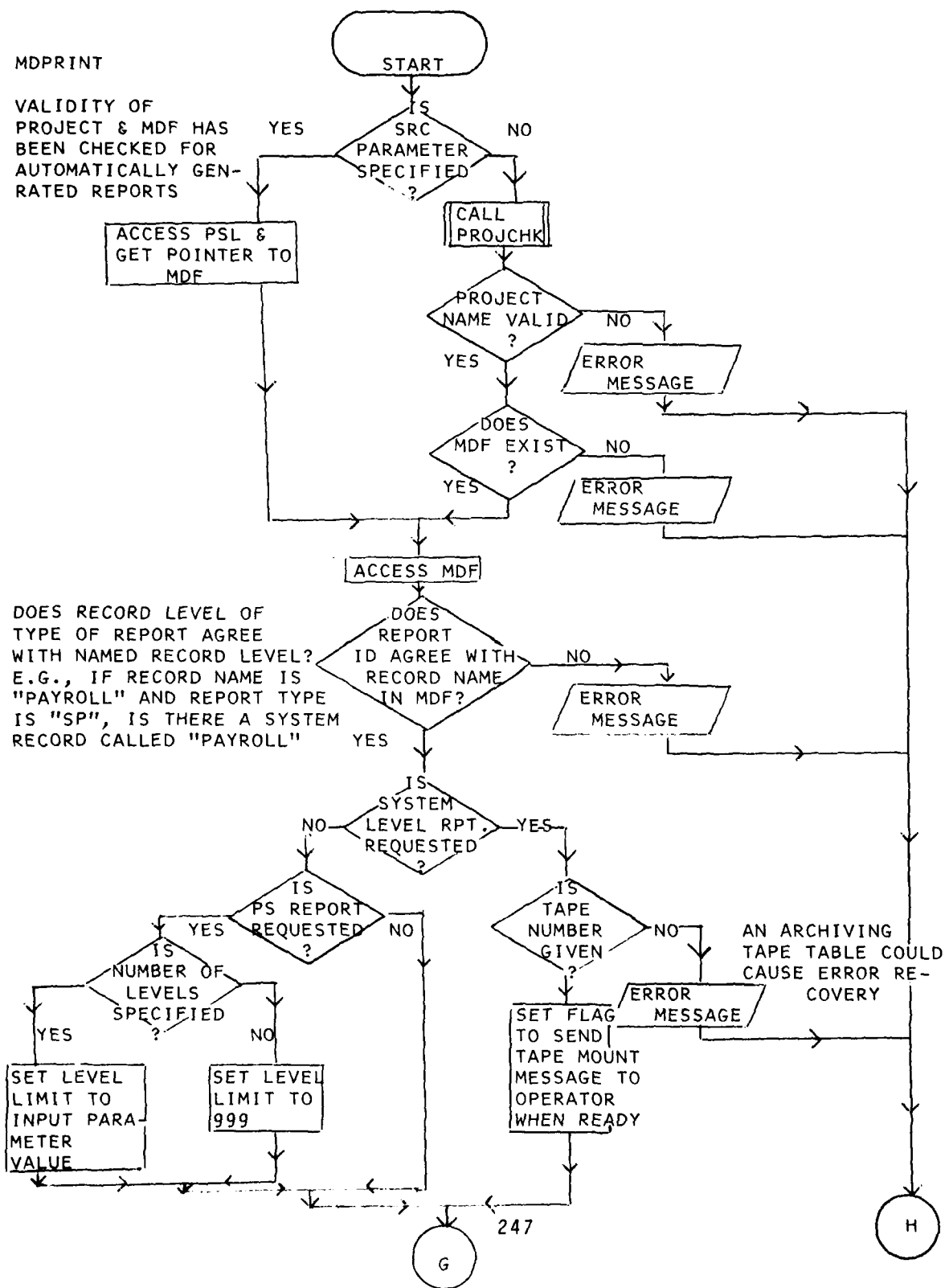
FO = parameter specifies the print format of the report. Possible values will be determined when the print programs are actually developed. It is suggested that the default value, assumed when the parameter is unspecified, be a standard format as shown in Figures 3.6-2 through 3.6-7.

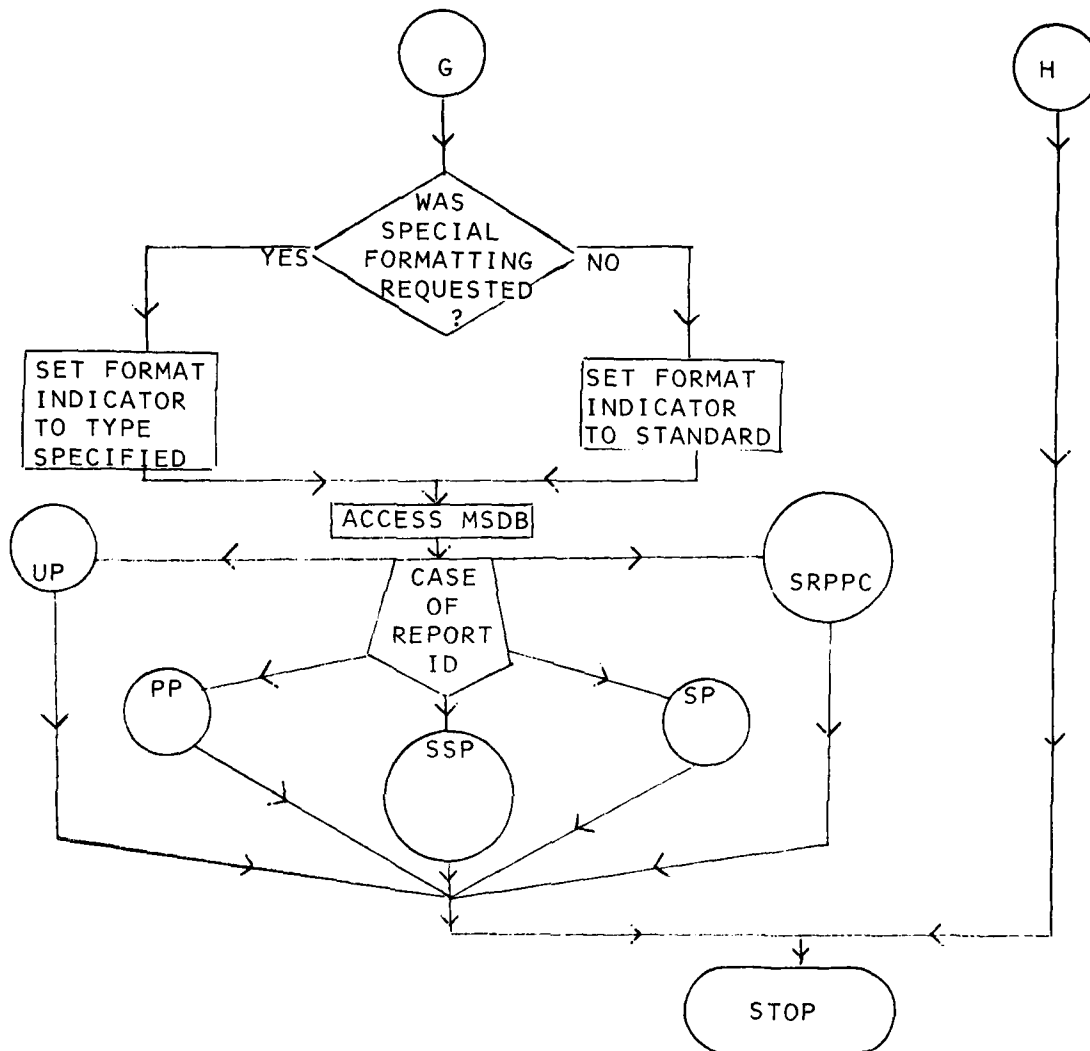
- Input:
- 1) project name
 - 2) report mnemonic
 - 3) record name of top level record
 - 4) number of levels if SP report or tape number if system level report
 - 5) source of report initiation is other than user
 - 6) format of printout

Output: Report printed in desired format.

MDPRINT

VALIDITY OF
PROJECT & MDF HAS
BEEN CHECKED FOR
AUTOMATICALLY GEN-
ERATED REPORTS





3.6.3.2.7 MDHIST

MDHIST has the responsibility of producing a Historical Report on the basis of information retrieved from the archived system level reports. This Historical Report, a sample of which may be found in SPS IX, Figure 3.6-6, allows the user to obtain a summary of the most pertinent data on development of a project over a specified time period. The exact contents of the report are listed in Section 6.0 of Appendix B, SPS IX.

Command format is:

MDHIST start date end date tape id FO = format

where:

start date is the beginning date of the time period of interest.

end date is the closing date of the time period of interest.

tape id is the identification of the tape volume(s) assigned for archiving the project (see MDPRI NT specifications, Section 3.6.3.2.6).

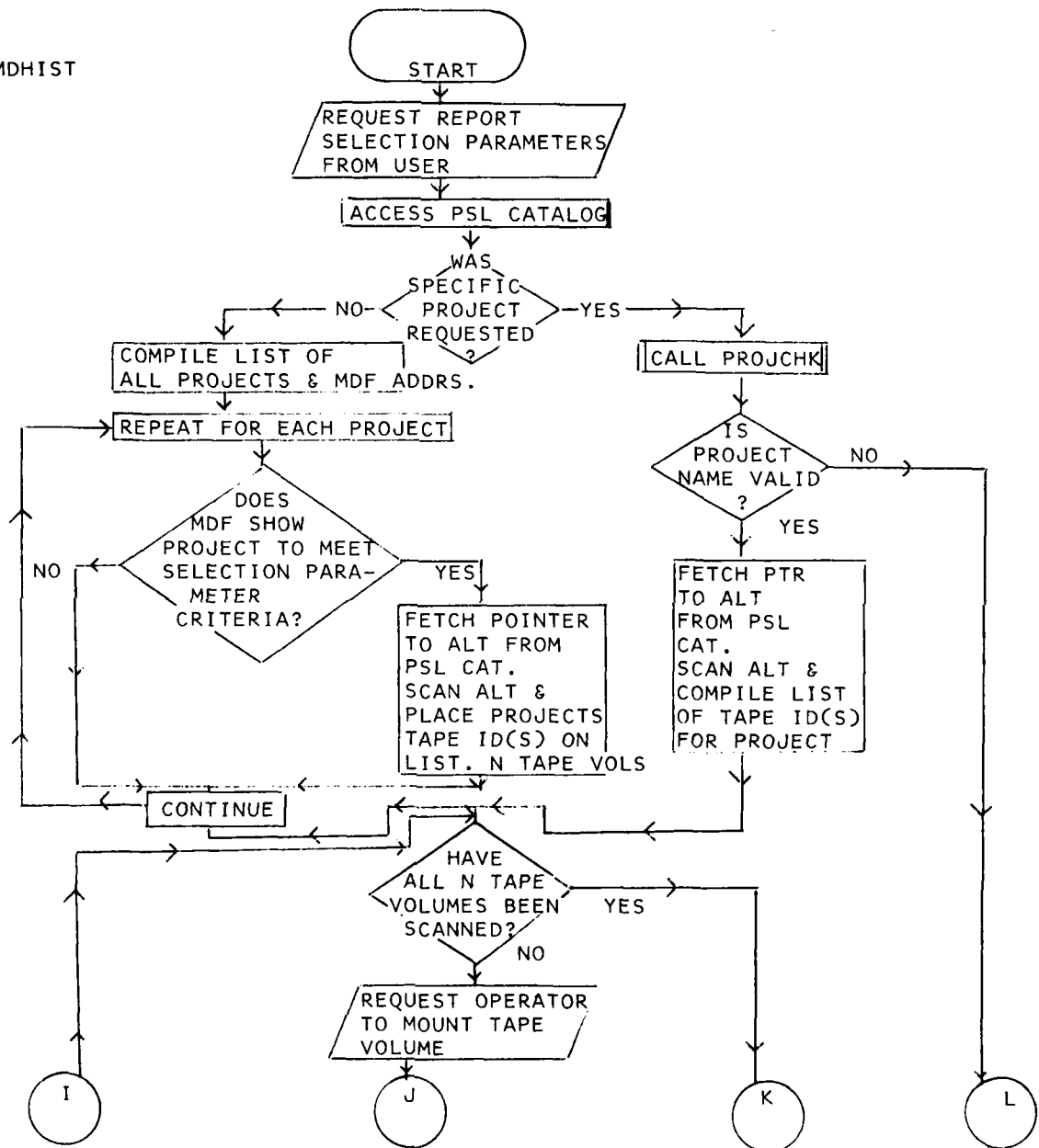
FO = format specifies the print format of the report. This parameter is further described in Section 3.6.3.2.6.

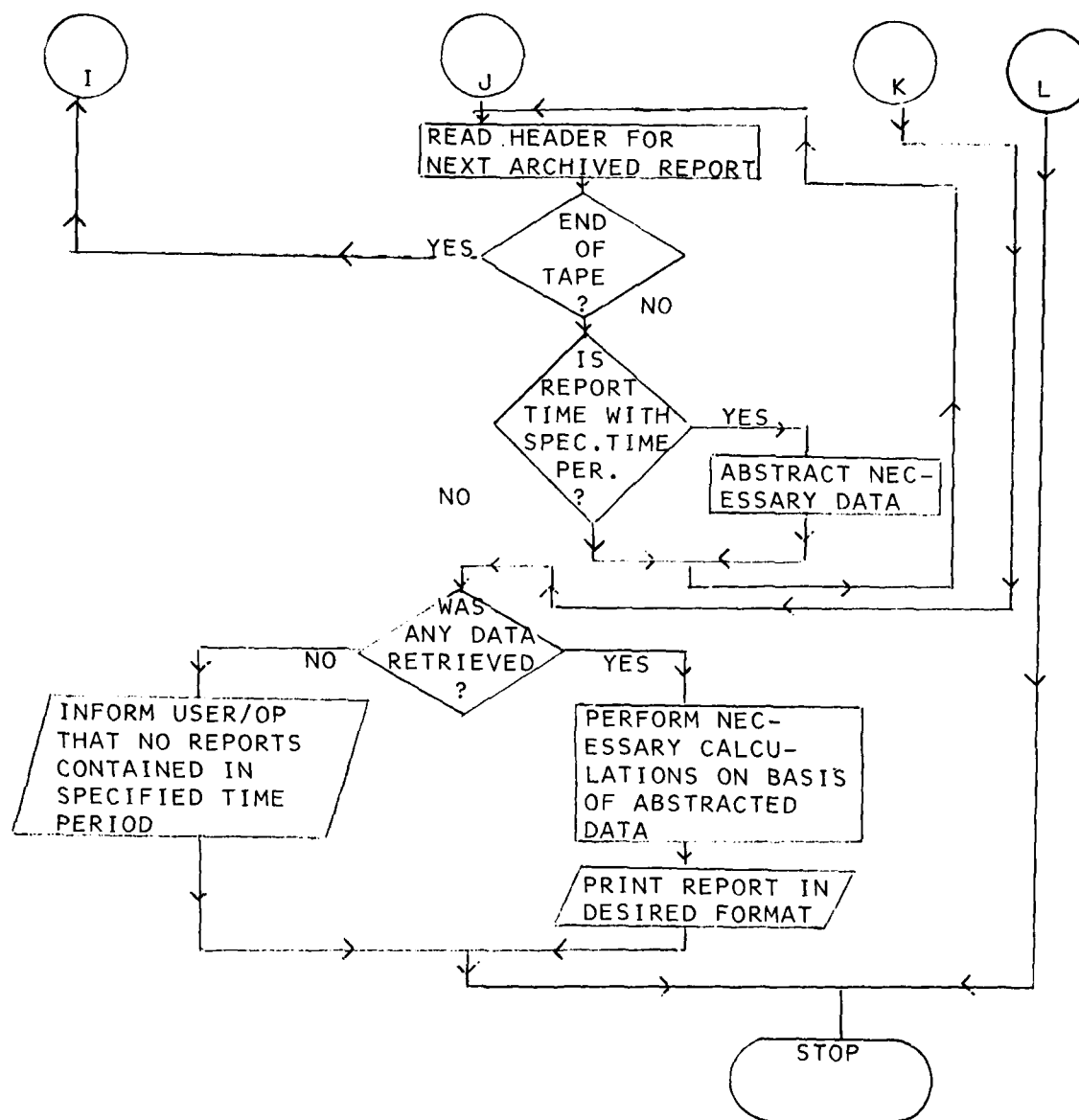
The SPS does not include in its data structures a location table of archived reports, i.e., an Archive Location Table (ALT). While such a table would undoubtedly be kept in the external library, it would be advantageous to keep it in the internal library as well. That would allow MDAUTORPT, MDPRI NT and MDHIST to verify and/or generate tape identification for system and historical reports. It would also allow the user to determine the correct tape id parameter value while still on-line. The pointer to this table could be placed in the PSL catalogue next to the pointer to the Name Table (See SPS VI, Figure 5.2).

- Input:
- 1) Starting and ending dates of report time period.
 - 2) Tape number(s) of archives for given project(s)
 - 3) Format of report

Output: Printed Historical Report in desired format.

MDHIST





3.6.3.2.8 MDXCHECK

MDXCHECK is responsible for establishing and maintaining the Exception Report Table which enables automatic generation of reports when specified MDF item values exceed allowed variances. The format of the ER Table may be found in Figure 3.6.3-4. That table is scanned daily by MDAUTORPT, as described in Section 3.6.3.2.5. The command format of MDXCHECK is:

MDXCHECK project name

where "project name" is the name of the report for which an exception report is to be generated.

The functional procedures for this program are the same as for MDREPORT, except that the Exception Table is the subject of examination. This would require that item identifiers and their variance values be requested from the user and checked for validity before entering them into or clearing them from the table. Since the major portion of this program is so similar to MDREPORT the flowcharts will not be repeated in this subsection. The reader is advised to reference subsection 3.6.3.2.4 of this report.

3.7 Documentation Support

Requirement:

This functional area involves the storage, update and output of program documentation. The requirement defined in this subsection are based on the use of a program design language, structured code and text as the principal sources of program documentation. Support of the generation, maintenance and output of graphics (e.g., block diagrams, HIPO charts, etc.) are not included in these requirements. The intent is to use the PSL storage and maintenance capabilities along with some specialized output capabilities to provide a basic documentation capability. It is not the intent of these requirements to use the PSL as a sophisticated document generation and publication system.

Purpose of Paragraph:

To define the scope of the Documentation Support functional area, and to emphasize that the requirements are intended to describe a support facility rather than a publication facility.

3.7.1 Basic Documentation Support Requirements

Classification: Basic

Requirement:

The storage and update of program design language, program source code and textual data are described in Subsection 3.1 of this report. The requirements defined in Subsection 3.1 are sufficient to cover all of the requirements for storing and updating program documentation. Additional documentation support is not included in the PSL basic requirements.

Purpose of Paragraph:

To define the scope of the Basic Documentation Support requirements and to reference that portion of the report that has previously described fulfillment of said requirements.

Examples:

SPS V, Subsection 3.1 concerns Source Data Maintenance. The paragraph included in that subsection are covered in Chapter 2 of this report, wherein several examples of storage and update of files (whether source, text, or PDL) are given. The reader is referred to that chapter for illustrations of Basic

Documentation Support implementation.

3.7.2 Full Documentation Support Requirements

3.7.2.a

Classification: Full

Requirement:

Full Documentation Support implementation includes the capability of printing in a user specified sequence one or more PSL files. This output facility must be able to merge Program Design Language files, source code files and textual data files into a single output listing.

Purpose of Paragraph:

To specify the need to print multiple files in a single listing.

Examples:

Example 3.7.2.a.1: Output of multiple sequenced files using a Type I system.

Type I facilities usually have a basic file manipulation utility which is capable of handling printing of several files in a single listing. The Hewlett-Packard RTE-IV operating system utilizes FMP, while DEC's RT-11 equivalent is PIP. They both operate in a similar manner, by allowing the user to direct listings to a device (printer in this case), then specify those files which are to be printed. An example of the PIP command fulfilling this function is given in Figure 3.7.2.a-1, which shows how three files with file name B55CMA and types of PDL, source, and text may be "copied" onto the line printer (LP:).

```
.R PIP
*LP:=B55CMA.PDL,B55CMA.FOR,B55CMA.TEX
*!C
```

Figure 3.7.2.a-1. Printing a specified sequence of files on a Type I system.

Example 3.7.2.a.2: Output of Multiple Sequenced Files
using a Type II System.

Both Type II systems investigated offer means of satisfying this requirement. CMS allows the user to spool printout on a continuous basis until otherwise notified with the command "SPOOL PRINT CONT". The actual printout is initiated by the PRINT command issued for each file to be printed. CSS achieves the same effect with the CLOSIO and OFFLINE PRINT commands, as shown below.

Figure 3.7.2.a.2-2 shows the EXEC file which, when executed, causes printout of the PDL, source and text files of a designated name. The CLOSIO command is first issued to turn off separation of files during printing. Next the title command is used to identify output. This is followed by the request to print a given file with the OFFLINE PRINT facility. The EXEC file terminates with the command to mark the end of the printed listing. Note that the EJECT option of the TITLE command may be used to start printout of files A on new pages. If no titles were desired between files, it would be necessary to use the OFFLINE PRINTCC version of the command, and to modify the first character of each record in the files to represent the desired carriage control instruction.

Actual execution of the EXEC file is accomplished by typing

DOCUMNT filename

where filename is the name of the file to be printed. It is assumed that that name is used on at least three files, one each of types PDL, SOURCE, and TEXTUAL.

If the user were to desire, a different print sequence, the CLOSIO and OFFLINE PRINT commands could be issued directly with appropriate parameters.

```
13.12.46 >PRINTF DOCUMNT EXEC
```

```
&TYPE OFF  
CLOSIO PRINTER OFF  
TITLE &1 PDL &DATE  
OFFLINE PRINT &1 PDL  
TITLE &1 SOURCE &DATE (EJECT)  
OFFLINE PRINT &1 SOURCE  
TITLE &1 TEXTUAL &DATE  
OFFLINE PRINT &1 TEXTUAL  
CLOSIO PRINTER ON
```

```
13.14.22 >
```

Figure 3.7.2.a-2. Printing a user-specified sequence of files from a PSL on a Type II system.

Example 3.7.2.a.3: Output of Multiple Sequenced Files using a Type III System.

The sample Type III system examined allows the user to output multiple files in a specified sequence as a single listing only by first copying the files into a single print file. Figure 3.7.2.a-3 shows the method for doing this.

First, the files which are to be printed must be made local files via the ATTACH command. Each file is then copied onto a single file in the desired order. The COPYSBF command is used to copy and simultaneously shift the data records one column to the right, thus moving a blank into the printer control column. This file may be edited to change the printer control characters to cause separation of segments of printout. Actual printing of this newly created file is accomplished with the BATCH command which routes the new file named 0 to the printer.

```
COMMAND- ATTACH, PDL, SAMPLEPDL, ID=GAERTNER, CY=1
COMMAND- ATTACH, S, SAMPLESOURCE, ID=GAERTNER, CY=3
COMMAND- ATTACH, TX, SAMPLETEXTUAL, ID=GAERTNER, CY=2
COMMAND- COPYSBF, PDL, 0
COMMAND- COPYSBF, S, 0
COMMAND- COPYSBF, TX, 0
COMMAND- BATCH, 0, PRINT, GAER
FILE NAME-IGAERB3 , DISP=PRINT , ID=**
COMMAND-
```

Figure 3.7.2.a-3. Printing a user specified sequence of files on a Type III system.

3.7.2.b

Classification: Full

Requirement:

Full requirements include the capability to format an output listing with the following user-specified information.

1. Page beginning indicators.
2. Header information to be printed at the top of each page.
3. Spacing between files being printed.
4. Spacing between lines of output.
5. Number of lines to print on a page.

Purpose of Paragraph:

To specify the need to allow user-defined formatting of printed output.

Examples:

Example 3.7.2.b.1: Formatting Output Listings using a Type I System.

None of the Type I systems investigated offer facilities to fulfill this requirement. However, it is not difficult to develop the software for output formatting. One small Government contractor developed the program LIST, shown in Figure 3.7.2.b-1, to format PDP-11 printout. It was written to satisfy the particular needs of the contractor, and so does not perform all the operations specified by Paragraph 3.7.2.b. However, modification to include the remaining features would not be difficult. As it exists, LIST's input parameters are file name, output device, lines per page, and textual notation. The latter consists of a maximum 60 character note which appears as the second header line on each page. The first header line includes the file name, time, date, and page number.

```

C      LIST : LIST A FILE
C      PRECEDING IT WITH FILE NAME, DATE, TIME, PAGE
C      AND A TITLE
C
C      IMPLICIT INTEGER (A-Z)
C      LOGICAL*1 FILNAM(13)
C      LOGICAL *1 ILAT(9), ITIME(9), CHSP, NULL, TAP
C      LOGICAL*1 AC(72), TC(60)
C
C      DATA CHSP/14 /, NULL/"2"/, TAP/"11"/
C
C      CALL ASSIGN(4, 'TT:/C')
C      CALL ASSIGN(7, 'AC:/C')
C      NLINES=55
C      CALL DATE(I, L, D)
C      CALL TIME(I, T, S)
C
C      IF I=10
C      THEN 7
C      10  FORMAT(' INPUT FILE')
C      11  IF(AC(5,1)) FILNAM
C      11  FORMAT(13A1)
C      11  DO 12 I=1,13
C      11  IF(FILNAM(I).NE.CHSP) GOTO 12
C      11  FILNAM(I)=NULL
C      11  GOTO 13
C      12  CONTINUE
C      13  CONTINUE
C      13  CALL ASSIGN(1, FILNAM, 2)
C
C      TYPE 5
C      REWIND 7
C      5  FORMAT(' OUTPUT UNIT')
C      5  ACCEPT 6, ICLDEV
C      6  FORMAT(I1)
C      6  IF(ICLDEV.EQ.3) GOTO 140
C      6  IF(ICLDEV.NE.7) GOTO 1
C      6  GOTO 14
C      140 CONTINUE
C      140 CALL ASSIGN(7)
C      140 CALL ASSIGN(7, 'KT:/C')
C      140 ICLDEV=7
C      14  TYPE 15
C      14  REWIND 7
C      15  FORMAT(' LINES PER PAGE')
C      15  ACCEPT 16, ILINES
C      16  FORMAT(I3)
C      16  IF(ILINES.LT.7) GOTO 14
C      16  IF(ILINES.NE.7) NLINES=ILINES
C
C      TYPE 20
C      REWIND 7
C      20  FORMAT(' TITLE?')
C      20  ACCEPT 30, 1
C      30  FORMAT(60A1)

```

Figure 3.7.2.b-1. Formatting Output listings using a Type I system. 258

```

      LC 35 TL=CC,1,-1
      IF(TC(TL).NE.C4SP) GOTO 36
35      CONTINUE
36      CONTINUE
C      NUMBER OF PAGES TO SKIP
      TYPE 40
      FFWIND 7
40      FORMAT(' PAGE NO. TO START WITH?')
      ACCEPT 16,NPSKIP
      IF(NPSKIP.LE.0) GOTO 33
      GOTO 33
C
44      NPSKIP=0
C
48      F=0
52      F=F+1
C
      IF(F.GE.NPSKIP)
100      IWRITE(IDEV,100)FILNAM,ILAT,ITIM,F,(ICD),I=1,TL)
      FORMAT(50/),T5,13A1,14A,9A1,15B,3A1,16B,'PAGE',13/,1X,(PA1)
      IF(F.GE.NPSKIP) FFWIND IDEV
      IF(F.GE.NPSKIP)
101      IWRITE(IDEV,101)
      IF(F.GE.NPSKIP) FFWIND IDEV
      FORMAT((/))
      LC 700 L=1,NLINES
C
      REAR(1,200,END=970)A
200      FORMAT(7PA1)
C
      DONT PRINT TRAILING BLANKS
      LC 500 C=70,1,-1
      IF(A(C).NE.C4SP) GOTO 600
500      CONTINUE
C
600      LC 510 C=1,C
510      IF(A(CC).EQ.TA1) A(CC)=C4SP
      IF(F.GE.NPSKIP)
      IWRITE(IDEV,610)(A(CC),CC=1,C)
610      FORMAT(1X,7PA1)
      IF(F.GE.NPSKIP) FFWIND IDEV
C
700      CONTINUE
      GOTO 30
800      TYPE 1000
      FFWIND 7
1000     FORMAT(30/)
      ENL

```

Figure 3.7.2.b-1 (continued). Formatting Output listings using a Type I system.

Example 3.7.2.b.2: Formatting Output Listings using a Type II System.

Both commercial time-sharing systems offer a choice of means for formatting printout. The simplest method is to use a text formatting program, SCRIPT, which can be supported by both CSS and CMS. To format output, the user would request normal, unformatted output and route it to his own disk. This he could then store as a single SCRIPT file and edit to insert the desired SCRIPT commands. The command set includes directives to allow user-specification of the five items listed in Paragraph 3.7.2.b, as well as a wide variety of additional functional capabilities.

Aside from this text formatter, CSS and CMS offer facilities to accomplish a majority of the required format functions. The CSS OFFLINE PRINT command normally produced printout which includes a heading consisting of file name, user i.d., date, time and page number. Spacing between lines and between items can be controlled if the output file has been modified to use column one for carriage control information. In this case, OFFLINE PRINTCC would be the correct command. CSS does not have the capability to let the user specify number of lines per page or where a new page should begin other than by carriage control information within the file. Figure 3.7.2.b-2 shows how this may be done, while Figure 3.7.2.b-3 gives the actual printout obtained.

CMS does allow the user to specify number of lines per page in the PRINT command. However, this parameter can only be used if the CC (carriage control) option is not in effect, thus giving the user the default spacing. If the CC option is invoked, the file is considered to include carriage control characters in column one. The CMS header is not as informative, giving only file name, and page number.

```
21.18.49 >printf sample text

OTHS IS A SAMPLE FILE
OIT WILL BE DOULBE SPACED
OEXCEPT FOR THE LINE FOLOWING THIS LINE
  A NEW PAGE WILL BEGIN WITH THE NEXT LINE
1THIS IS A NEW PAGE

21.19.08 >offline Printcc sample text

21.19.30 >
```

Figure 3.7.2.b-2. Formatting output listings using a Type II system.

THIS IS A SAMPLE FILE
IT WILL BE DOUBLE SPACED
EXCEPT FOR THE LINE FOLLOWING THIS LINE
A NEW PAGE WILL BEGIN WITH THE NEXT LINE

THIS IS A NEW PAGE

Figure 3.7.2.b-3. Formatted output listing on a Type II system

Example 3.7.2.b.3: Formatting Output Listings Using
a Type III System.

Insertion of carriage control characters is the only means the user has of formatting the printout of the CDC 6600. The printer assumes that the file is preformatted, thus necessitating the use of the COPYSBF command for files not previously formatted. This command was shown in Example 3.7.2.a.3, which the reader should reference.

3.7.2.c

Classification: Full

Requirement:

Complete formatting facilities must include automatic page numbering beginning with a user-supplied page number or with page 1.

Purpose of Paragraph:

To describe the required page numbering capability.

Examples:

None of the systems examined had standard facilities which let the user specify the number of the first page of printout. The Type I and III systems do not number output pages at all, unless output is pre-processed by a program such as LIST (see Example 3.7.2.b.1). The Type II systems number printed pages beginning with 1, but can accept user-defined page numbers if a text formatting program has been purchased and linked to the system. The following example shows the use of the SCRIPT facility to alter page numbering.

Example 3.7.2.c-1: Use of a Text Formatting Program in a Type II system.

Figure 3.7.2.c-1 shows how the ".pp" SCRIPT command can be inserted in a file to cause printout to begin at the page number specified. Since SCRIPT also automatically formats each line of a file, it is necessary to turn the formatting option off (".nf") if the data is to be printed exactly as listed. Note that the file containing such commands must have a file type of SCRIPT.

```
18.26.42 >printf sample script
```

```
.ps 32
.nf
1.3   4.2
7.4   7.1
2.2   7.4
```

```
18.26.49 >runoff sample (offline)
```

```
18.27.05 >
```

```
1.3   4.2
7.4   7.1
2.2   7.4
```

Figure 3.7.2.c-1. Use of a Text Formatting Program in a Type II system.

3.7.2.d

Classification: Full

Requirement:

Full Documentation Support requires the capability to print a title page which contains as a minimum the following user-specified information:

1. Document Title.
2. Document Date.
3. Name(s) of Author(s).
4. Organization and Address.

Purpose of Paragraph:

To specify the need for and content of a title page for printout.

Examples:

Neither the Type I or the Type III systems have the capability specified in this paragraph. Additional software would be required for either facility to print a title page.

Example 3.7.2.d.1: Title page generation in a Type II System.

The Type II systems examined can print title pages, either via use of SCRIPT as previously shown or (for CSS only) by use of the TITLE command. This command causes a user-specified heading to appear on the first two pages of the printed output that is requested following command issuance. The heading consists of three eight character fields, which may contain this paragraph's specifications 1 through 3 if the instruction is given as follows:

TITLE title date name.

CSS automatically includes organization name and address at the start of each listing. This information is kept in a special address file the contents of which are specified when the user contracts to use the system. Typical printout of organization identification is shown in Figure 3.7.2.d-1.

MAIL FIRST CLASS TO : WMS
W.W. CAERTNER RESEARCH INC.
1492 HIGH RIDGE RD. 2ND FLOOR, RM 1
STAMFORD, CT. 06903

Figure 3.7.2.d-1. Example of organization name and address printed by a Type II system.

3.7.2.e

Classification: Full

Requirement:

Full Documentation Support includes the capability to generate a magnetic tape in print image form. This allows the distribution of documentation in machine readable form, thus facilitating both storage and reproduction of documentation.

Purpose of Paragraph:

To state the need for transferrable machine-readable documentation.

Examples:

Type I and III Systems do not have the capability to create tapes in print image format, and would require additional software in order to do so.

Example 3.7.2.e.1: Generation of Magnetic Tape in Print Image Format in a Type II system.

The commercial time share systems both have facilities for generating such tapes. CMS offers the TAPE utility, while CSS utilizes the program UTILITY. As shown in Figure 3.7.2.e-1, the desired listings are stored on the user's disk, then transferred to tape via the DT feature of UTILITY. Material stored on the magnetic tape thus produced can be printed at any computer installation which supports a compatible utility program. The actual printing is accomplished

with the TL tape-to-listing command or its equivalent.

17.49.22 >mount tape scratch as tap2 ringin

17.50.04 >sleep

FROM IDDP : OK, PLS HOLD

DEV TAP2 ATTACHED

17.51.21 >printf prog1 fortran

C SAMPLE PROGRAM

C

READ(5,1000) N,M

1000 FORMAT(2I4)

L = N ** M

C

WRITE(6,1001) N,M,L

1001 FORMAT(' ',I4,' ** ',I4,' = ',I10)

STOP

END

17.51.34 >utility

REQUEST

>dt

INPUT FILE?

>prog1 fortran F

BLOCKING FACTOR?

>

OUTPUT TAPE?

>tap2

REQUEST

>css

CSS:

>tape rewind tap2

>

REQUEST

>t1

LRECL?

>80

FILES?

>prog1 fortran F

FILES?

>

INPUT TAPE?

>tap2

EOF 0

REQUEST

>quit

17.53.02 >

PRU00010
PRU00020
PRU00030
PRU00040
PRU00050
PRU00060
PRU00070
PRU00080
PRU00090
PRU00100

C SAMPLE PROGRAM

C

READ(5,1000) N,M

1000 FORMAT(2I4)

L = N ** M

C WRITE(6,1001) N,M,L

1001 FORMAT(' ',I4,' ** ',I4,' = ',I10)

STOP

END

Figure 3.7.2.e-1. Generation of magnetic tape print image format in a Type II system.

3.7.3 Comparative Analysis of Documentation Support Facilities

Of the three types of systems investigated, only commercial time sharing facilities are able to offer a suitable environment for documentation support. Assuming that the facilities use a text-formatting program such as SCRIPT, all requirements for both Basic and Full Documentation Support can be met.

This is not true of the other two system types. The small in-house computer can come close to meeting the Basic requirements (see Chapter 2 of this report), but completely lacks the software to fulfill the Full requirements. The Type III system cannot comply with the requirements at either the Basic or Full level, and is totally unsuited for text processing.

Since so many of the available systems lack Full Documentation Support facilities, this report will include specifications for software to fulfill the Full documentation requirements. The following program design is intended for an interactive environment, but can readily be adapted for batch.

The program is initiated by the command LISTING, which has the format

LISTING	<u>PRINT</u>	SAVE
	<u>TAPE</u>	<u>NO SAVE</u>

where:

PRINT specifies that the output is to be routed to the printer. This is the default.

TAPE specifies that the output is to be routed to the tape drive to be designated.

SAVE specifies that the print file created is to be saved on disk under the designated name. This option is only valid if PRINT was specified as the first parameter.

NOSAVE specifies that the print file is to be erased after transfer to printer. This is the default.

Figure 3.7.3-1 shows a sample dialogue resulting from use of this command. That dialogue illustrates the prompts, required input, and functional performance of LISTING PRINT NOSAVE. The limitations on input values is discussed below in item 1); items 2) and 3) concern the LISTING command when the TAPE and SAVE parameters are used.

1) LISTING PRINT NOSAVE (System Default)

- a) Output files to be printed in a single listing are each specified by file name, file type, subdivision and project name. Each file designation is terminated by a "/". A null entry terminates execution of the program.
- b) Spacing between lines of printout are given as an integer, n , $1 \leq n \leq 3$ to denote single, double or triple spacing. A null entry signifies the system default of 1.
- c) Spacing between files may be entered as an integer n , $1 \leq n \leq 10$ or as the character string "NP", designating "new page". If an integer, the value must be greater than or equal to the value specified for line spacing. If null, a default of twice the line spacing value is assumed.
- d) Number of lines per page must be an integer n , $1 \leq n \leq 66$. If null, default of 55 is assumed. If value exceeds 62, header is eliminated.
- e) Number of the first printed page following the title page must be an integer n , $1 \leq n \leq 2^7$. If null, the default of 1 is assumed.
- f) Each of the two heading lines consists of three twelve-character fields printed at the left, middle and right of the top two printable lines on each page. Each heading line is requested separately. Header fields are delimited by "/".
- g) Report title is a character string of up to 40 characters which is printed on the title page.

- h) Up to 10 authors of the report may be specified, each name being delimited by "/".
- i) The date of the document is obtained from the system.
- j) The name and address of the organization authorizing the listing is contained in a system user table which is account related. The information is stored in the system at the time the organization requests a computer account. This information is also printed out on the title page.
- k) Page blocking (i.e., the ability to specify the beginning of a new page) is also offered to the user. Note that an earlier parameter can cause printout of each file to start on a new page. Further page blocking is made possible by this parameter. If the user responds "yes", then he may also request a printout of the total print file as it is formatted at that point. The user then identifies the lines which are to begin new pages. A null entry signifies the termination of new page specification.

2) LISTING PRINT SAVE

These parameters result in essentially the same dialogue as the NOSAVE option, except that a print image of the listing is saved under a user-specified file name with a file type of PRINTLIST. A PRNTLIST file must be specified as input file for the TAPE parameter (see item 3).

3) LISTING TAPE

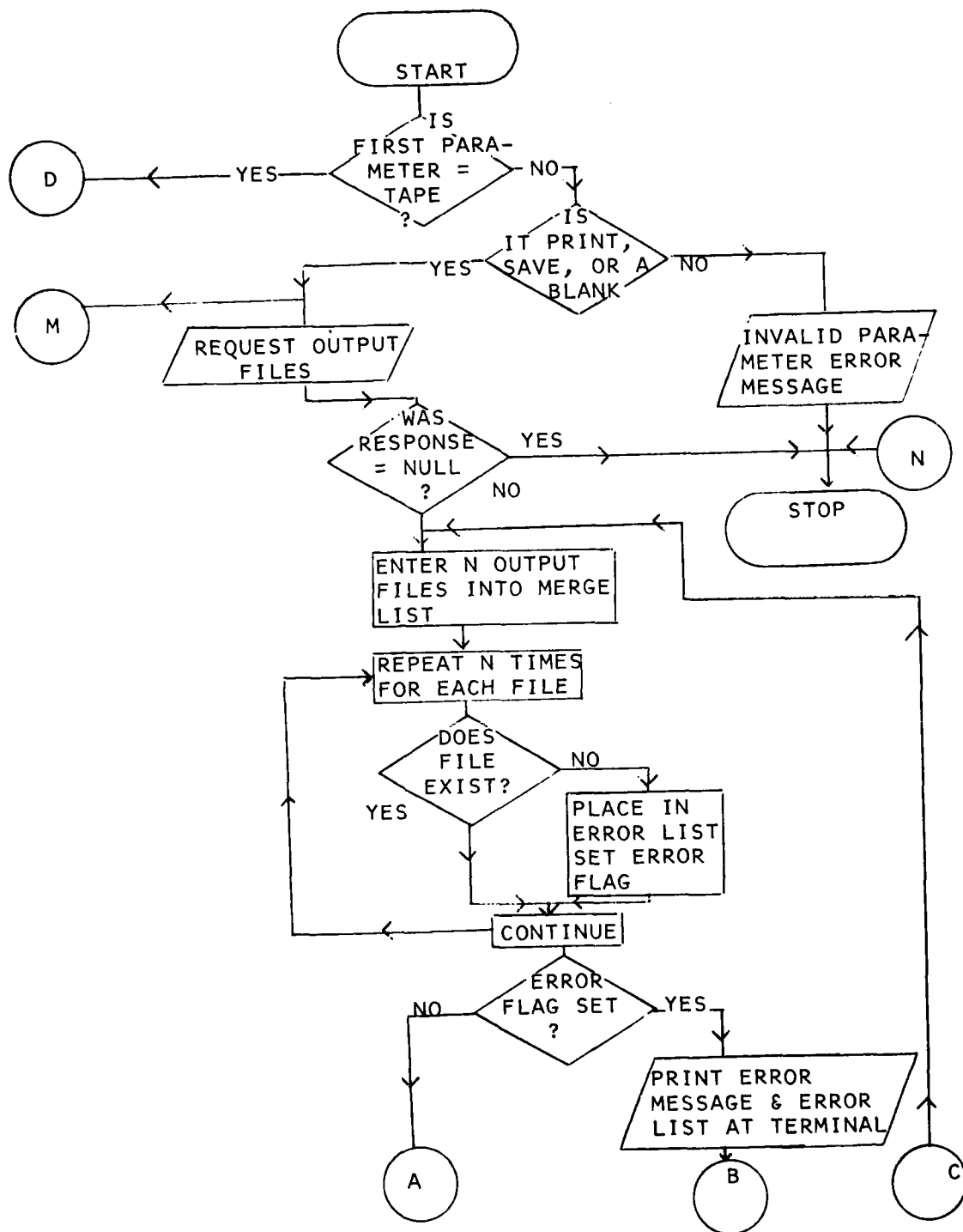
This allows the specified PRNTLIST file to be transferred to the specified tape volume.

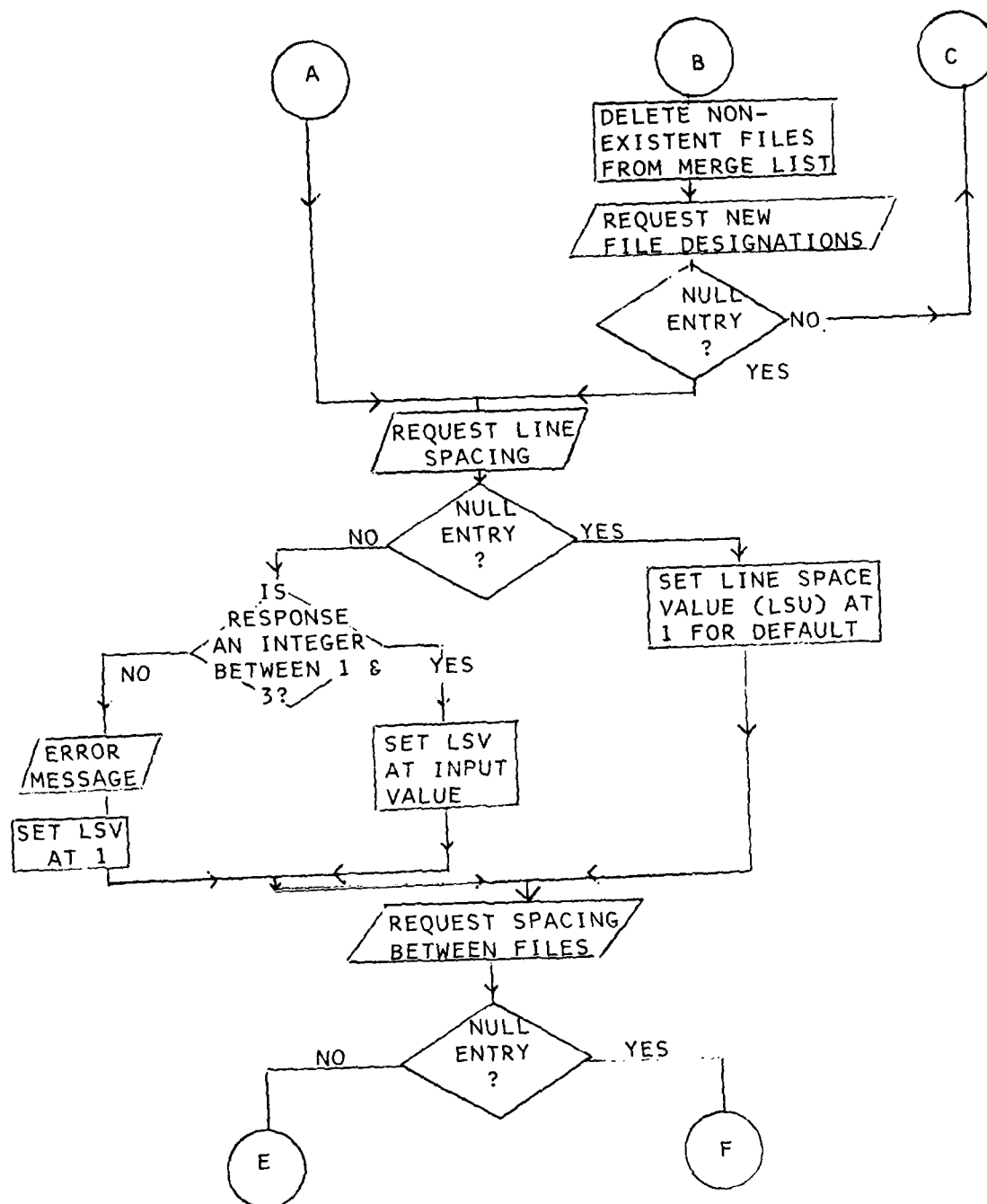
```

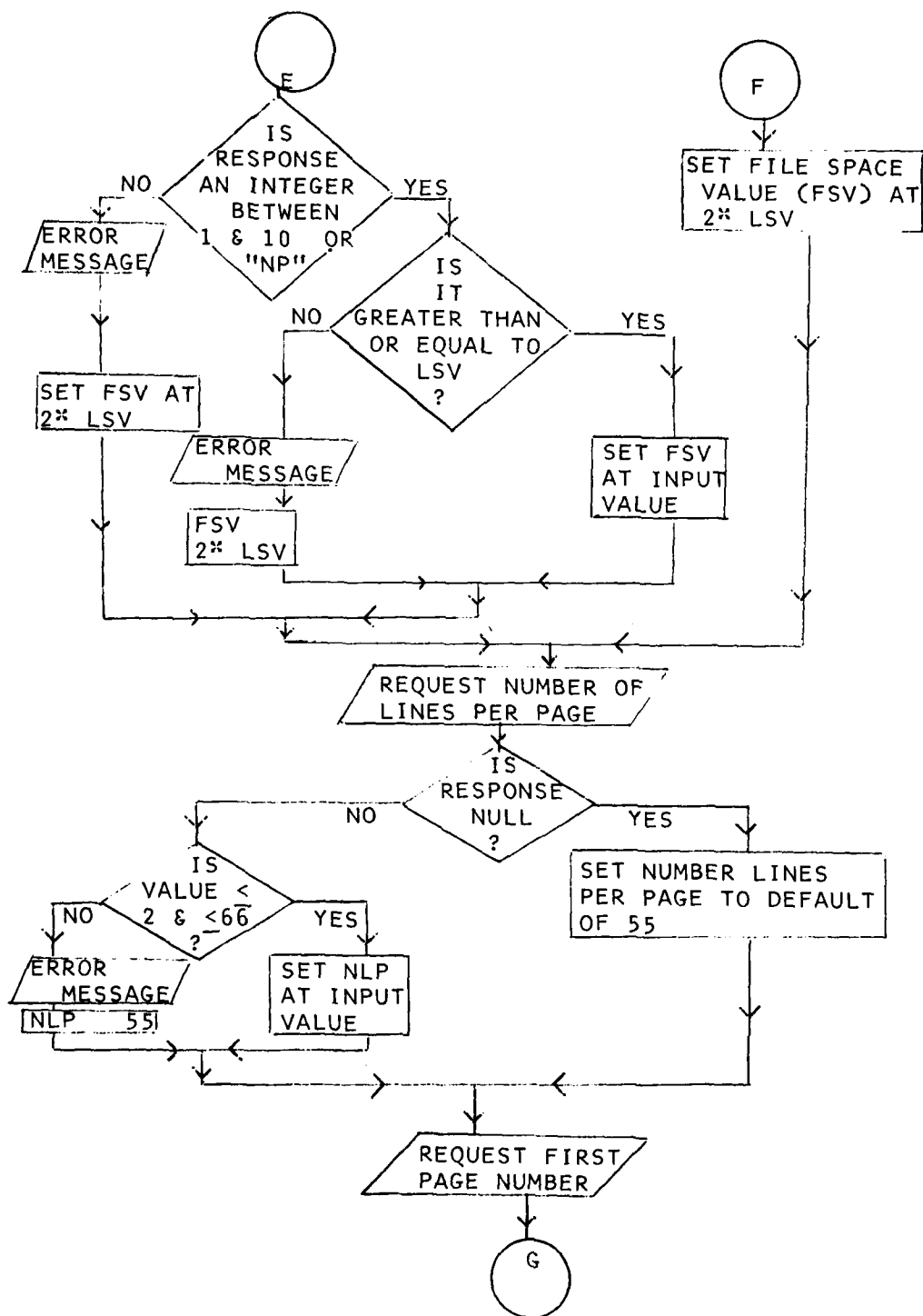
> LISTING PRINT NOSAVE
. ENTER OUTPUT FILES
  FILENAME  FILETYPE  SUB. PROJ/FILENAME FILETYPE SUB. PROJ./.../
. ENTER SPACING BETWEEN LINES
  n or null
. ENTER SPACING BETWEEN FILES
  n or null
. ENTER NUMBER LINES PER PAGE
  n or null
. ENTER 1ST PAGE NO.
  n or null
. ENTER HEADING LINE 1
  /string 1/string 2/string 3/or null
. ENTER HEADING LINE 2
  /string 1/string 2/string 3/or null
. ENTER REPORT TITLE
  n characters
. ENTER AUTHOR(S)
  Auth 1/Auth 2/Auth 3/.../
. FORMATTING...
. DO YOU WISH TO ALTER PAGE BLOCKING?
  yes or no
ENTER OUTPUT FILES
.
.
.

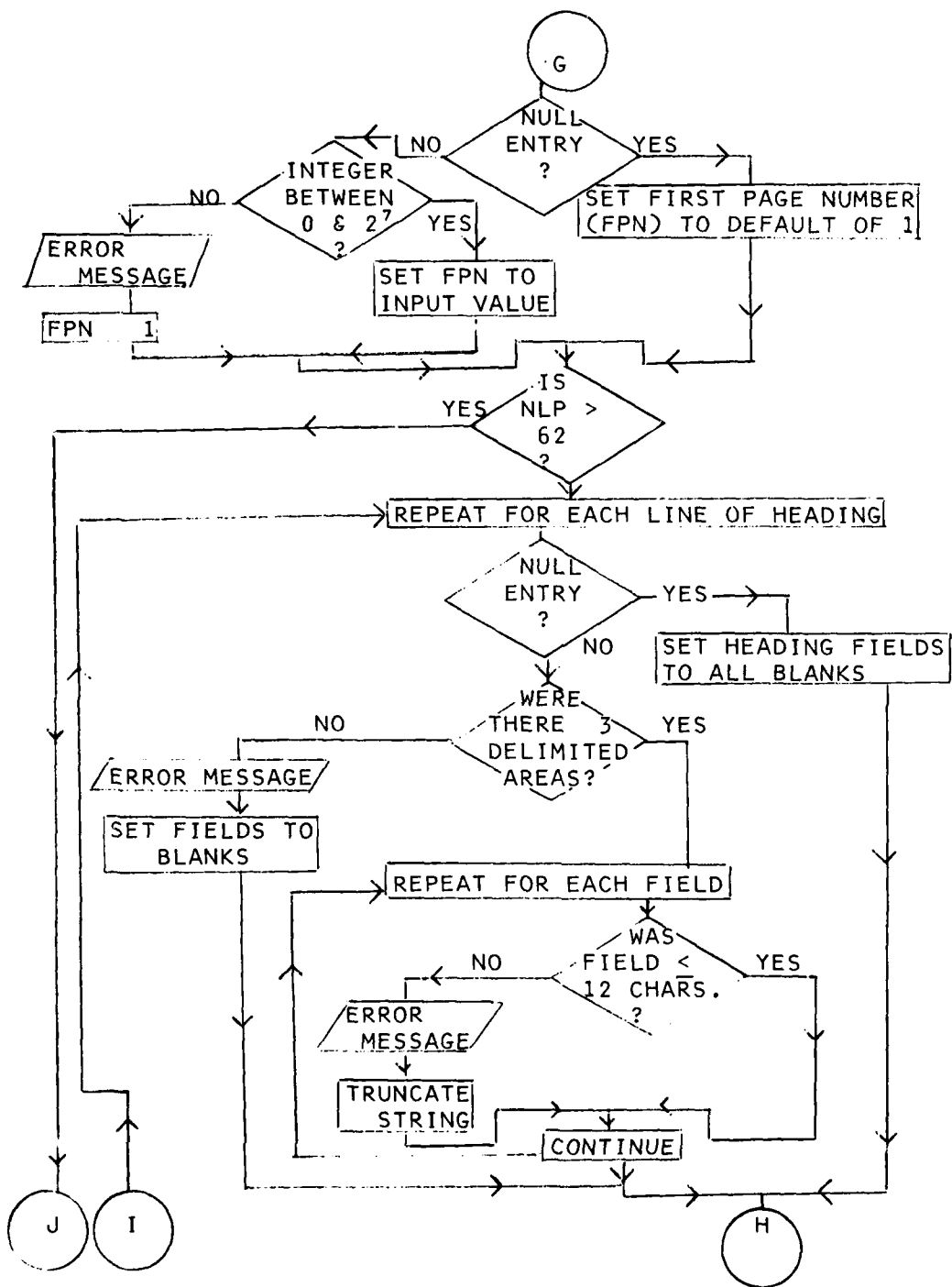
```

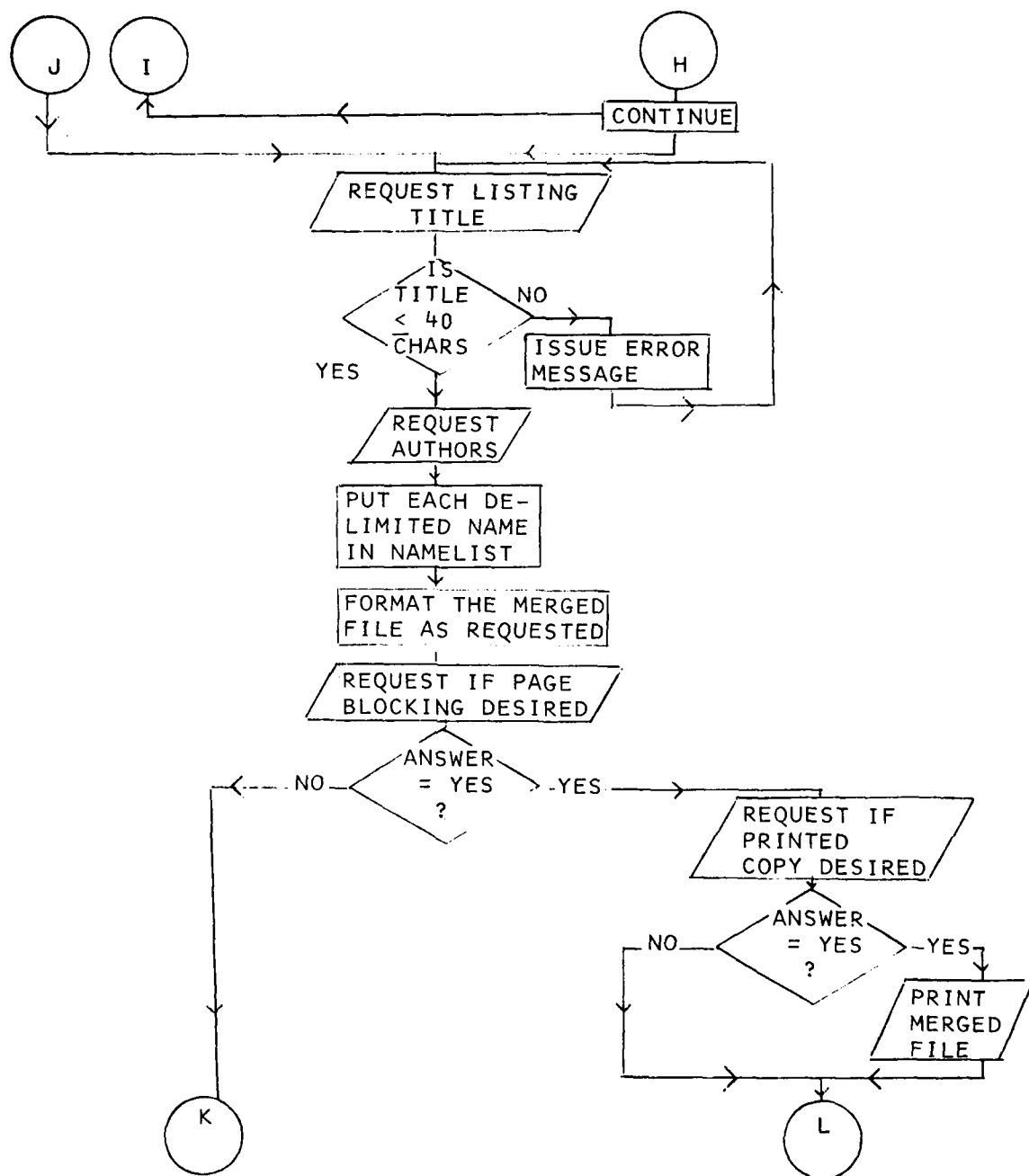
Figure 3.7.3-1. Sample dialogue of proposed LISTING command.

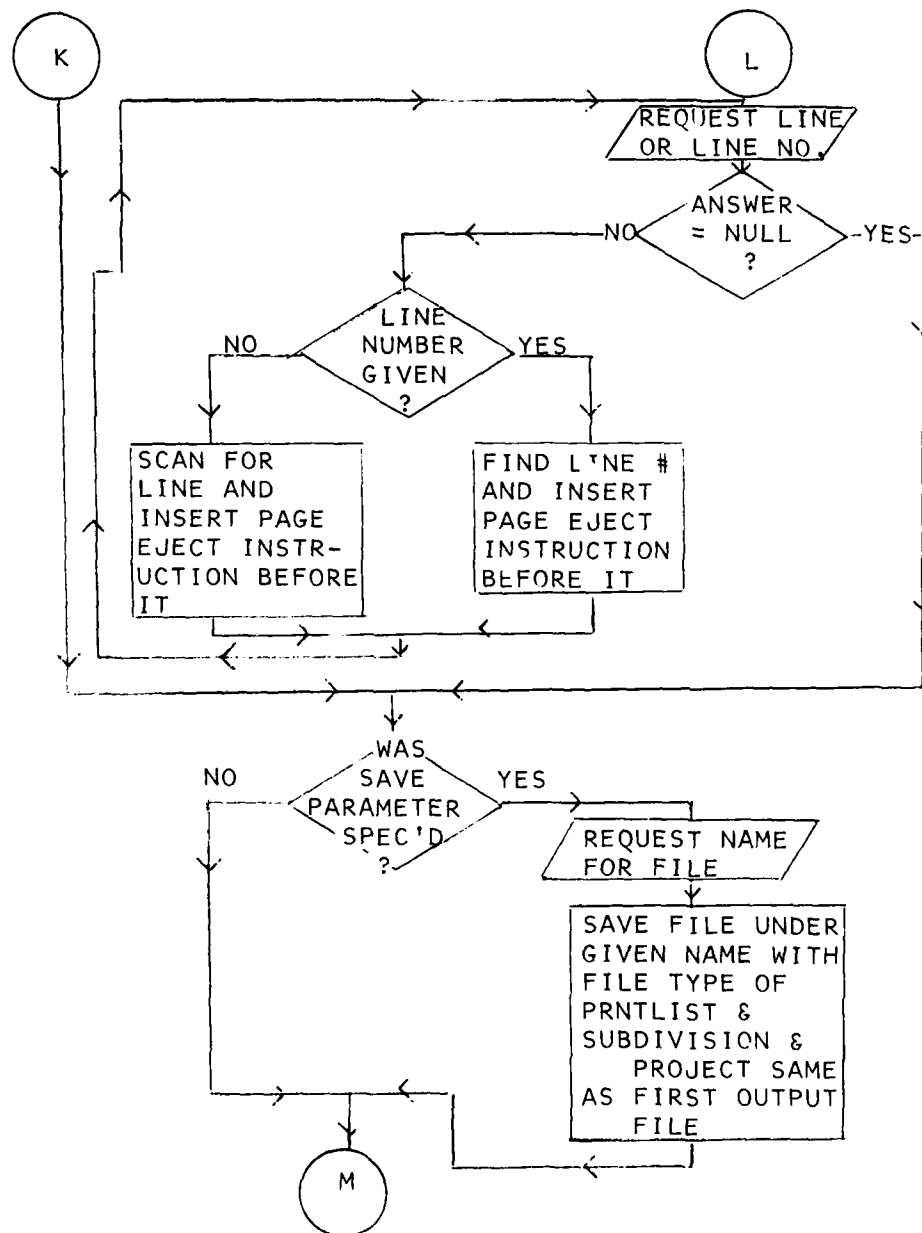


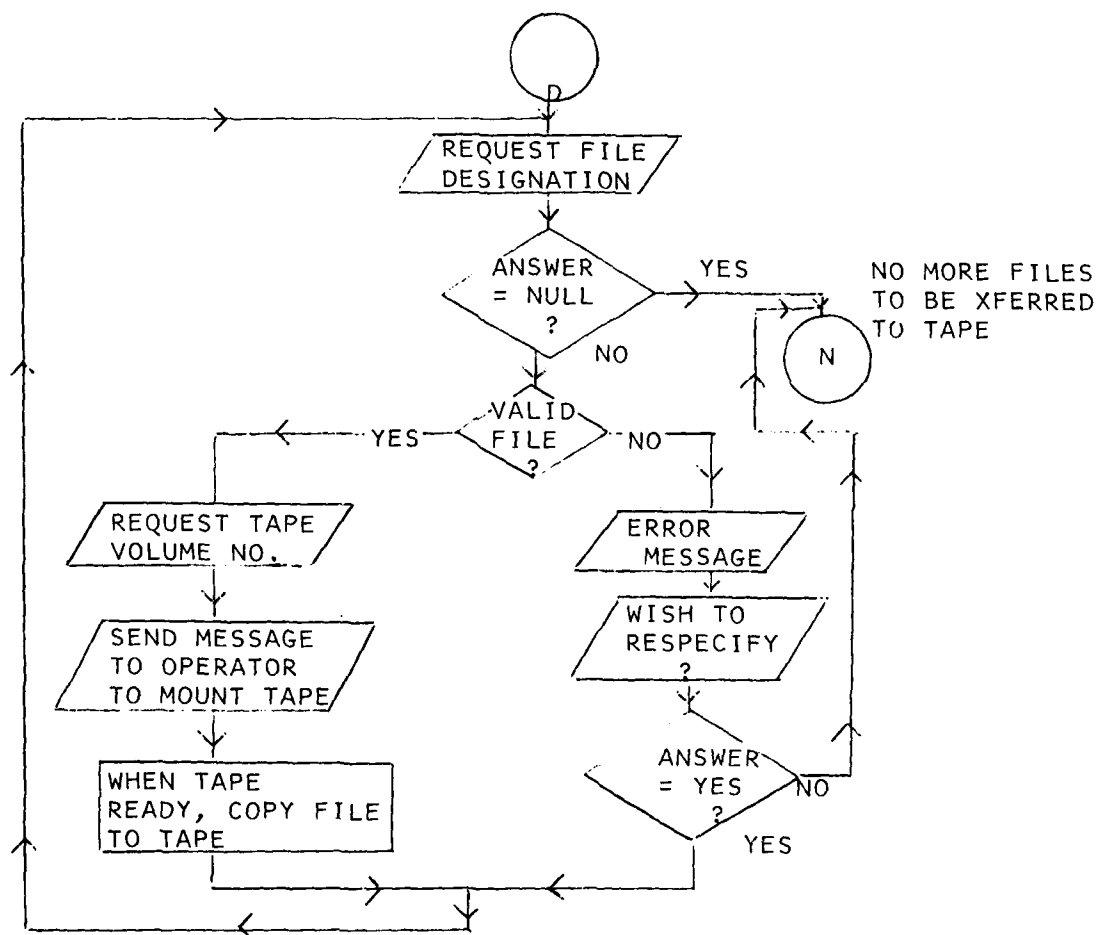












AD-A081 389

GAERTNER (W W) RESEARCH INC STAMFORD CONN
PROGRAMMING SUPPORT LIBRARY, VOLUME II. GUIDELINES FOR IMPLEMENTATION
NOV 79 C M TURCIO, W W SCHREYER, N A ADAMS F30602-78-C-0103

F/G 9/2

UNCLASSIFIED

RADC-TR-79-241-VOL-2

NL

4 x 4

AD-A081 389



END

DATE

FILED

4-80

DTIC

3.8 On-Line Terminal Implementation Requirements

Requirement:

Up to this point the descriptions of the PSL functional requirements have been independent of any implementation considerations. A system to satisfy these requirements could be implemented in either a batch or an on-line environment. However, the use of on-line terminals for the dynamic execution of normal library functions (e.g., source data maintenance, program compilation, etc.) introduces some additional requirements which apply only to an on-line implementation of a PSL. These requirements, which cover general on-line terminal capabilities, are stated in this subsection.

An on-line implementation of a PSL must provide the following capabilities to support the functional requirements defined in the other subsections of this report.

Purpose of Paragraph:

To introduce the special requirements which pertain to on-line implementation of the PSL.

Limitations of Examples Within This Area:

The requirements discussed within this section are not so much dependent upon the type of system used - minicomputer, time sharing, or large computer - as upon the hardware which has been purchased for that installation and the operating system chosen to support it. Since the variety of hardware available is both large and ever-increasing, the reader should realize that the following examples are just a small sampling of possibilities. If interested in interactive operations, the individual contractor is urged to thoroughly investigate all hardware compatible with his system in order to find that which is best suited to his needs.

3.8.1 Basic On-Line Terminal Implementation Requirements

3.8.1.a

Classification: Basic

Requirement:

On-line implementation of a PSL must include support of terminals which are connected to the computer either via communication links or directly via cables and control units (if required).

Purpose of Paragraph:

To state the types of terminal-to-computer linkages required for on-line implementation of a PSL.

Examples:

Example 3.8.1.a.1: Terminal-to-Computer Linkage in Type I, II and III Systems.

Since Type I systems are designed to be used in-house, the typical terminal linkage uses cable. However, most operating systems available for minicomputers with memories larger than 16K can also support long distance communication links.

Commercial time-sharing facilities (Type II) support both means of linkage. One of the systems investigated could be accessed via telephone from the United States, Canada, and Europe, and could support 700 concurrent users without appreciable system degradation. That same system also made remote batch facilities available upon request. The other time-sharing system was a university-based research installation which supported over 30 on-site (cable connected) terminals as well as numerous remote linkages with university branches throughout the state.

Type III systems are primarily batch facilities. The hardware and software necessary to support interactive operation have often been tacked on almost as an after-thought. These systems tend to offer the least-sophisticated interactive capabilities, yet most Type III systems do support both hard-

wired and dialup communication links, as well as remote batch facilities.

In summary, the three major system types all may be associated with operating systems and hardware which allow terminal-to-computer linkage both directly via cable and remotely via dialup links.

3.8.1.b

Classification: Basic

Requirement:

Interactive PSL capabilities must include support of both display (Cathode Ray Tube) terminals and typewriter terminals.

Purpose of Paragraph:

To identify the generic types of terminals which must be supported by an on-line PSL implementation.

Examples:

Example 3.8.1.b.1: Terminal Types Supported by Type I, II and III Systems.

Each system type can fulfill this requirement in that it is associated with at least one operating system and (if necessary) hardware extensions which allow both terminal types to be used.

Teletype terminals are often used on in-house minicomputers though more typewriter-like devices such as an IBM 2741 can also be attached. Type I systems also support CRT's with operating speeds ranging from 110 to 9600 baud. The only usual limitation on terminal type is that it should use ASCII or EBCDIC character codes.

Type II systems likewise can use a wide variety of ASCII and EBCDIC terminals, both CRTs and typewriter style. Some of the more recently developed CRTs have hardware features such as a character insert mode which facilitate text editing. Often the terminal operating speed is user-selectable, and can range from 10 CPS to 120 CPS.

The Type III systems also frequently offer user selectable terminal speeds. Terminals operating via long distance communication links can have speed determined by the access number dialed for link up. These terminals may be either CRT or typewriter style, but often are limited to being ASCII, as EBCDIC is not typically supported by the average computer used in a Type III System.

3.8.1.c

Classification: Basic

Requirement:

It is necessary to have a remote job entry capability so that PSL jobs initiated at an on-line terminal may be run in a batch environment.

Purpose of Paragraph:

To specify the need for a remote job entry facility.

Examples:

Example 3.8.1.c.1: Remote Job Entry on a Type I system.

Most but not all Type I systems allow the user to create a job stream for execution in batch mode. A contractor will have to determine if the operating system in use supports this facility. When it does exist, RJE usually takes the form of scheduling execution of files which contain job control language commands. Figure 3.8.1.c-1 shows such a file, as it would be executed to process a job in the RT-11 BATCH facility. This file, SAMPLE.BAT, allows a FORTRAN file, SAMPLE.FOR, to be created and stored on disk DK1. The file contents will also be listed on the assigned LST device since the LIST option is given in the \$CREATE command. Following file creation, the FORTRAN code is compiled, linked, and executed, and the file printed on the list device (\$FORTRAN/RUN/LIST). This command also allows the object and load modules to be saved under designated file names. The use of the BANNER and TIME options in the \$JOB command causes a header and the time of execution of each command to be printed in the log device.

Execution of SAMPLE.BAT is shown in Fig. 3.8.1.c-2. First the BATCH run-time handler and log device must be made resident with a LOAD command, then the LOG and LST devices assigned. In the example, the line printer (LP) is used for both LST and LOG, though a terminal could also have been specified as either device. Finally, the BATCH compiler is run with the .R BATCH command, and the desired batch file executed. More than one file could have been specified if necessary.

Although the main purpose of BATCH is to allow programs to run unattended, most batching facilities also allow communication with the system operator, thus creating a highly flexible batch environment capable of fulfilling a variety of user/job needs.

```
$JOB/BANNER/TIME
$MESSAGE THE FOLLOWING IS A SAMPLE BATCH
$MESSAGE STREAM FOR THE CREATION, COMPILATION,
$MESSAGE AND EXECUTION OF A FORTRAN PROGRAM.
$CREAT/LIST DK: SAMPLE.FOR
      I = 1
      J = 2
      K = I + J
      WRITE (6,1) K
1     FORMAT (' THE VALUE OF K IS ',I2)
      STOP
      END
$EOD
$FORTRAN/RUN/LIST DK: SAMPLE.FOR/SOURCE -
                DK: SAMPLE.OBJ/OBJECT SAMPLE.SAV/EXECUTE
$EOJ
```

Figure 3.8.1.c-1. Batch File in a Type I System.

```
.LOAD BA,LP
.ASSIGN LP LOG
.ASSIGN LP LST

.RUN BATCH
*SAMPLE.BAT

END BATCH
```

Figure 3.8.1.c-2. Execution of a Batch File in a Type I system.

Example 3.8.1.c.2: Remote Job Entry on a Type II System.

Commercial time-sharing systems offer two alternatives for RJE: Connection of a remote batch terminal to the computer or creation of a job stream to be executed in batch mode. The first method requires additional hardware which is linked to the host computer via standard communication linkage procedures. The second choice does not necessitate additional hardware, and so is used more often.

The job stream which is created by the user can, in some systems, be associated with a time option which specifies when the job is to be executed. This allows the user to take advantage of lower rates available in non-peak hours. The job stream itself takes the form of standard system commands, since batch facilities for a virtual computer usually consist of an individual virtual machine designated specifically as a batch processor.

The example below shows how batch jobs are entered in CMS. Three Exec files (Figures 3.8.1.c-3 through 3.8.1.c-5) are created which allow the user to assemble Assembler Language source files batch mode simply by issuing a single command.¹ If a user with ID GAERTNER wished to assemble the file ACCOUNTS, he would type

```
BATCH GAERTNER ACCOUNTS ASSEMBLE.
```

The system would then respond

```
PUN FILE nnnn TO BATCHCMS  
R;
```

where nnnn is the sequential number assigned to the punch file. The BATCH command causes execution of the three EXEC files, which function to punch the source file to the batch reader, to link the batch machine to the user's disk for access of the necessary macro library and to actually assemble the designated file. Other EXEC files could be set up to perform additional batch functions such as program execution or compilation in various high-level language.

¹ IBM VM/370: EXEC User's Guide, pp. 58-59.

```

1.  BATCH EXEC
*   THIS EXEC SUBMITS ASSEMBLIES/COMPILATIONS TO CMS BATCH
*
*   -PUNCH BATCH JOB CARD:
*   -CALL INPUT EXEC TO PUNCH DATA FILE;
*   -CALL APPROPRIATE LANG. EXEC ($3) TO PUNCH EXECUTABLE COMMANDS
*
& CONTROL ERROR
& IF INDEX GT 2 $ SKIP 2
& TYPE CORRECT FORM IS:  BATCH USER ID FNAME FTYPE (LANG.)
& EXIT 100
& ERROR $ GOTO -ERR1
CP SPOOL D CONT TO BATCH CMS
& PUNCH /JOB $1 1111 &1
EXEC INPUT &1 &2
EXEC &3 &2 &1
& PUNCH /*
CP SPOOL D NO CONT
CP CLOSE D
CP SPOOL D OFF
& EXIT
-ERR 1 &EXIT 100

```

Figure 3.8.1.c-3. Batch Exec Control Program to allow RJE in a Type II system.

```

2.  INPUT EXEC
*   CORRECT FORM IS:  INPUT FNAME FTYPE
*   PUNCH DATA FILE FOR BATCH PROCESSOR;
*   TPE /* LINE BEHIND THE DATA FILE IS
*   TRANSLATED TO A NULL LINE BY BATCH
*   SO THAT MOVEFILE RECOGNIZES THE END
*   OF THE DATA SET
*
&CONTROL ERROR
&ERROR &GOTO -ERR3
&PUNCH FILEDEF INMOVE TERM (BLOCK 80 CRECL RECFM F
&PUNCH FILEDEF OUTMOVE DISK &1 &2 (BLOCK 80 CRECL 80 RECFM F
&PUNCH FILEMOVE
&PUNCH &1 &2 * (NO HEADER)
&PUNCH /*
&EXIT
-ERR &EXIT 103

```

Figure 3.8.1.c-4. INPUT EXEC Required by BATCH for RJE in a Type II system.

3. ASSEMBLE EXEC

```
* CORRECT FORM IS: ASSEMBLE FNAME USERID
*
* PUNCH COMMANDS TO:
*   -INVOKE CMS ASSEMBLER
*   -RETURN TEXT DECK TO CALLER
&CONTROL ERROR
&ERROR &GOTO -ERR2
&PUNCH CP LINK $2 191 199 RR PASS= RPASS
&BEGPUNCH
ACCESS 199 B/B
GLOBAL MACLIB VPLIB CMSLIB OSMACRO
RELEASE 199
&END
&PUNCH CP MSG &2 ASMBLING '&1'
&PUNCH ASSEMBLE &1 (PRINT NOTEROM)
&PUNCH CP MSG &2 ASSEMBLY DONE
&PUNCH CP SPOOL D TO &2 NOCONT
&PUNCH PUNCH &1 TEST A1 (NO HEADER)
&BEGPUNCH
CP CLOSE D
CP SPOOL D OFF
CP DETACH 199
&END
&EXIT
-ERR2 &EXIT 102
```

Figure 3.8.1.c-5. EXEC causing assembly via RJE in a Type II system

Example 3.8.1.c-3: Remote Job Entry on a Type III system.

Much like the Type II systems, Type III allows RJE either via the special hardware of a remote batch terminal or via a specific interactive command to transmit a batch job. The latter method is shown in Figure 3.8.1.c-6.

The command is BATCH, and files specified for processing must contain all necessary job control cards. The INPUT subcommand sends the designated file to the input queue at the central site, thus if the file is to be compiled, executed, etc., the proper JCL must be added in via the Editor. The word "HERE" following the INPUT subcommand routes the output back to the user's terminal output queue. This is later made accessible to examination via the LOCAL disposition.

The PRINT command causes the named file to be printed at the central site, while card punching is done by the PUNCH command. If a file is to be punched in binary, the PUNCHB command may be used. It should be noted that any access of magnetic tape on this system requires operating in batch mode, as previously discussed.

Like the commercial time-sharing system, the particular Type III system examined also allows the user to specify the priority of transmitted batch jobs. Thus the user may control the scheduling of job execution so as to take advantage of lower rates at non-peak times.

COMMAND-

```
      BATCH
TYPE FILE NAME- XEQSAMPLE
TYPE DISPOSITION- INPUT,HERE.
TYPE FILENAME- COMPILED LISTING
TYPE DISPOSITION- PRINT
TYPE FILE ID- GAER
TYPE FILE NAME- DOCUMENTATION TEXT BACKUP
TYPE DISPOSITION- PUNCH
TYPE FILE ID- WWGP
TYPE FILE NAME- XEQSAMPLE
TYPE FILE DISPOSITION- LOCAL
TYPE FILE NAME- END
COMMAND-
```

Figure 3.8.1.c-6. Dialogue of Remote Job Entry on a Type III system.

3.8.1.d

Classification: Basic

Requirement:

On-line implementation of a PSL requires a terminal-to-terminal communication capability to allow the transfer of data between interactive ports within the same PSL system.

Purpose of Paragraph:

To state the need for inter-terminal communication.

Examples:

Example 3.8.1.d.1: Inter-terminal Communication in a Type I System.

In-house minicomputer systems can vary widely in size from the 8K single user installation to the multi-user 500 K systems. Naturally, only the operating systems intended for use on a multi-terminal facility could include interterminal communication capabilities. For example, both RT-11 and RSTS may be used on PDP-11 machines, but the former does not allow transfer of files or short messages between terminals as it is completely unnecessary. RT-11 is designed to work in a single user environment. If more than one terminal is concurrently operational, they must both be accessing the same disk, thus obviating the need for data transfer.

RSTS and other operating systems geared for many users usually have means of transferring both files and short messages between terminals.

Example 3.8.1.d.2: Interterminal Communication on a Type II System.

Both commercial time-sharing systems investigated offer excellent inter-terminal communication facilities. Short messages between terminals may be sent with the MSG command, while actual transfer of data in CSS is accomplished with the XFER and OFFLINE PUNCH commands. The CMS equivalents are SPOOL PUNCH and PUNCH.

The sample dialogue in Figure 3.8.1.d-1 illustrates communication between virtual machines GAERTNER and GRI., as seen from GAERTNER's terminal. The user receives a request via MSG for the transfer of the file SAMPLE DATA. He responds by sending a confirming message, then transferring his card punch (device D) routing GRI. The requested file is punched, and finally the virtual card punch routing is returned to normal (XFER D OFF). The same operation as viewed from GRI's terminal is shown in Figure 3.8.1.d-2.

```
08.32.41 >
FROM GRI: PLS SEND SAMPLE DATA

08.33.06 >MSG GRI OK, WILL XFR

08.33.48 >XFER D TO GRI

08.34.03 >OFFLINE PUNCH SAMPLE DATA

08.34.29 >XFER D OFF

08.34.43 >MSG GRI DID U GET IT?

08.35.12 >
FROM GRI: YUP
```

Figure 3.8.1.d-1. Terminal-to-terminal communication capability of a Type II system showing both message and data file transfer.

```
08.32.23 >MSG GAERTNER PLS SEND SAMPLE DATA

08.33.48 >
FROM GAERTNER: OK, WILL XFER

CARDS TO BE READ

>OFFLINE READ *
OFFLINE READ SAMPLE DATA

08.34.33 >
FROM GAERTNER: DID U GET IT?

08.34.54 >MSG GAERTNER YUP
```

Figure 3.8.1.d-2. Terminal-to-terminal communication capability of a Type II system showing both message and data file receipt. 287

Example 3.8.1.d.3: Interterminal communication on a Type III system.

The representative Type III system studied has no need for a means of file transfer, as all files are stored on a common device and are accessible to any user who knows the filename and passwords (if any). Users at distant terminals may communicate to determine permanent file names (PFN) or other information via SEND command. This is illustrated in Figure 3.8.1.d-3 wherein a terminal user requests user MJEK to tell him the name of a file.

```
COMMAND- SEND  
  
TO WHOM - MJEK  
  
TYPE MESSAGE OR END-  
WHAT IS PFN OF COMPUTE A  
  
TYPE MESSAGE OR END-  
END  
  
COMMAND-  
FROM MJEK - PFN=B55CMPUTALPHA
```

Figure 3.8.1.d-3. Terminal-to-terminal communication on a Type III System showing message sending and receipt.

3.8.1.e

Classification: Basic

Requirement:

A facility to recover from system failure must be included in the PSL so that data which has been entered from a terminal prior to a hardware or software failure is not lost (i.e., a user may reinitialize a terminal session at the point of failure so that only the data displayed on the terminal but not entered into the system is lost).

Purpose of Paragraph:

To specify the need for a means to minimize loss of interactively entered data when a system failure occurs.

Examples:

Example 3.8.1.e.1: System Failure Recovery in a Type I System.

The Type I systems examined all offer salvage of everything but the operation in progress when a system failure occurs. Unfortunately, if the operation in progress is an editing session, a great deal of information can be lost. The means to avoid this when using the RT-11 is to exit from the editor at regular intervals, thus saving all work up to that point (See Fig. 3.8.1.e-1). Other operating systems offer an Editor SAVE command which saves the session work done since last save or editor initiation.

```
.R E
*EBSAMPLE.DAT$R$$
*
.
. (EDIT COMMANDS)
.
*EX$$
.R E
*EBSAMPLE.DAT$R$$
*
.
. (EDIT COMMANDS)
.
*EX$$
.R E
*EBSAMPLE.DAT$R$$
*
.
. (EDIT COMMANDS)
.
*EX$$
```

Figure 3.8.1.e-1. Ensuring minimum loss of data while performing editor commands on a Type I system.

Example 3.8.1.e.2: System Failure Recovery in a Type II System.

Like the Type I systems, only editing sessions make the commercial time-sharing user vulnerable to data loss. However, most Type II systems have a SAVE command which allows the editing session thus far to be saved without exiting the editor. Fig. 3.8.1.e-2 shows the CSS version of the SAVE command as used during an editing session in which a stub is replaced with actual code. First the stub is deleted, then two magnetic cards (maximum 50 lines each) are used to input the code. The SAVE command is given following the presentation of the first magnetic card. The FILE command which terminates the editing session also saves the editing session.

CSS and CMS further allow the user to specify a periodic save after every n file updates, where n is a user specified number. Thus if the user were to type "AUTOSAVE 30" at the beginning of the terminal session, he would see the message "SAVED" displayed at his terminal when a total of 30 additions, deletions, or changes have been made to the file since the last save message. The current setting of AUTOSAVE may be determined by typing "AUTOSAVE" without operands, while the automatic save facility may be turned off by typing "AUTOSAVE OFF".

```
09.04.21 >EDIT SAMPLE FORTRAN
EDIT:
>L /WRITE/
WRITE(I DIAG,1000)
>DE 99
EOF
>I
INPUT:
>      .
>      . (FIRST CARD OF 50 LINES IS INPUT)
>      .
> (CARRIAGE RETURN)
EDIT:
>SAVE
INPUT:
>      .
>      . (SECOND CARD IS ENTERED)
>      .
> (CARRIAGE RETURN)
EDIT:
>FILE
09.09.36 >
```

Figure 3.8.1.e-2. Ensuring minimum loss of data while inputting a file on a Type II system.

Example 3.8.1.e.3: System Failure Recovery on a Type III System.

Although the Type III system offers a SAVE command, this unfortunately does not create a file copy which can be salvaged after system failure. In order to do this, the local file created by the SAVE command must be cataloged as a permanent file (see Fig. 3.8.1.e-3. Note that the temporary file created by cataloging the intermediate edit results must be purged if special efficiency is to be maintained.

```
COMMAND- ATTACH,X,SAMPLEFORT,ID=GAERTNER

PF CYCLE NO.= 001
COMMAND- EDITOR
..E,X,S

..      .
..      .
..      . (EDIT COMMANDS)
..      .
..S,XT,N

..CATALOG,XT,TEMP,ID=GAERTNER

INITIAL CATALOG
RP = 090 DAYS
CT ID= GAERTNER PFN=TEMP
CT CY= 001      0000312 WORDS.:
..      .
..      .
..      . (EDIT COMMANDS)
..      .
..S,XF,N

..BYE

COMMAND- CATALOG,XF,SAMPLEFORT,ID=GAERTNER

NEW CYCLE CATALOG
RP = 090 DAYS
CT ID= GAERTNER PFN=SAMPLEFORT
CT CY= 002      0000312 WORDS.:
COMMAND- PURGE,XT

PR ID= GAERTNER PFN=TEMP
PR CY= 001      0000312 WORDS.:
COMMAND-
```

Figure 3.8.1.e-3. Ensuring minimum loss of data during edit session on a Type III system.

3.8.1.f

Classification: Basic

Requirement:

An on-line PSL implementation requires the capability to scan through data files displayed at the terminal. This may be termed a paging capability for those terminals which are page-oriented, such as CRTs, but even those terminals which are not page-oriented should be able to display the first, last, and specified intermediate lines of a file. The total file scanning capability includes (where applicable):

1. Displaying the next page of data. A page of data is defined as the number of lines which may be displayed on the CRT at a single time; this number is dependent upon the specific make of CRT.
2. Displaying the previous page of data.
3. Displaying the first page of data retrieved.
4. Displaying the last page of data retrieved.
5. Rolling a display forward one or more lines.
6. Rolling a display backward one or more lines.

Purpose of Paragraph:

To define the file display capabilities necessary at an on-line terminal.

Existing Tools Which Satisfy This Requirement:

Most interactive editors contain the capability to perform functions specified in subparagraphs 3. through 6. Page oriented CRTs either have hardware paging devices, or paging commands, or can use the roll-back and roll-forward capabilities to display the previous (next) page.

Examples:

Most of these capabilities were discussed in Chapter 2's discussion of SPS V Paragraph 3.1.1.d, particularly the section on Functional Capabilities of Editors. They will be briefly reviewed in this section.

Example 3.8.1.f.1: On-line File Scanning Capabilities
in Type I Systems.

The minicomputer systems examined do not offer software paging facilities, but do allow full file scanning via editor commands. This is illustrated in Figure 3.8.1.f-1. Since this particular dialogue was generated on a typewriter terminal, the line being pointed to at a given time is displayed via the "V" command. The actual pointer positioning is done as follows:

- 1) "17A" to advance the pointer to the first line of the second page. This assumes 18 lines per display page, if the terminal were a CRT. In that case, the "V" command would not be needed.) "-17A" would advance backward one page.)
- 2) "A" to advance the pointer to the next line. A positive or negative integer may precede the command to specify the number of lines forward or backward which the pointer will be moved.
- 3) "/A" to advance the pointer to the last line of the file.
- 4) "B" to return the pointer to the first line of the file.

Note that the above assumes that the entire file is small enough to fit into the Text Buffer. If such is not the case, then the current buffer must be copied onto the output file and the next buffer contents read in from the report file via the "N" command.

```
.R EDIT
*EBSAMPLE.DAT$ISV$$
THIS IS THE FIRST LINE OF THE FILE
*17A$V$$
THIS IS THE FIRST LINE OF PAGE TWO
*A$V$$
THIS IS THE SECOND LINE OF PAGE TWO
*-2A$V$$
THIS IS THE LAST LINE OF PAGE ONE
*/A$V$$
THIS IS THE LAST LINE OF THE FILE
*B$V$$
THIS IS THE FIRST LINE OF THE FILE
*TC
```

Figure 3.8.1.f-1. On-line file scanning capabilities in a Type I system.

Example 3.8.1.f.2: On-Line File Scanning Capabilities in a Type II system.

The commercial time-sharing editors offer capabilities so operationally similar to Type I systems that no dialogue will be given. The only differences between the two systems lie in the mnemonics and in the buffer. The text buffer of the Type II system is large enough to accommodate any file. The mnemonics are as follows:

- 1) TOP to position pointer to the top of file.
- 2) BOTTOM to position pointer to the end of the file.
- 3) NEXT or DOWN to advance the pointer. An integer may be specified after the command in order to advance multiple lines.
- 4) UP to move the pointer back a line. Again, an integer following the command will move the pointer back the designated number of lines.

Example 3.8.1.f.3: File Scanning Capabilities in a Type III System.

The Type III system studied offers a PAGE command which allows the user to perform all required functions except displaying the first and last pages of a file. The command is reviewed in Figure 3.8.1.f-3. It includes subcommands which allow the user to page forward and backward and to roll (by line) the screen image forward or backward the specified number of lines. The PAGE command does not allow the user to display the first or last page of a file other than by paging forward or backward until it is reached.

```
COMMAND- PAGE, SAMPLEDATA
READY...
+
+ [First 18 line page is displayed]
+ [Next 18 line page is displayed]
R12 [Screen display rolls forward 12 lines]
-R5 [Screen display rolls backward 5 lines]
R-2 [Screen display rolls backward 2 lines]
[Preceding 18 line page is displayed]
Q
COMMAND-
```

Figure 3.8.1.f-2. File Scanning in a Type III system.

3.8.2 Full On-Line Terminal Implementation Requirements

3.8.2.a

Classification: Full

Requirement:

Full interactive PSL implementations requires the capability to collect and output statistics related to each on-line terminal. Such statistics include time in use (real and CPU), types and number of requests, etc.

Purpose of Paragraph:

To specify the need for reporting statistics on terminal usage.

Examples:

Example 3.8.2.a.1: Terminal Use Statistic Reporting in a Type I System.

None of the Type I systems explored currently offer automatic reporting of information on terminal use. Certain data can be collected manually (e.g., hours on line), especially if a typewriter terminal is used. The session record can be saved and later analyzed to extract any desired statistics.

If a more automated statistics collection plan were desired, new software would have to be developed. Since such software would be highly dependent upon the type of system for which it is collecting data, only the computer manufacturer could properly supply it, or the user himself.

Example 3.8.2.a.2: Terminal Use Statistics Reporting in a Type II System.

One of the two commercial time-sharing systems (CSS) offers basic utilization reporting for accounting purposes, as well as allowing a complete record of each terminal session to be kept as a file. CMS also offers the latter capability, but the installation examined uses a more rudimentary accounting scheme.

Figure 3.8.2.a-1 contains a printout of typical statistics reported for billing purposes. Note that some minimal information is also given on types of requests, e.g., whether a type was accessed or printout obtained. No other analysis is done of

types of requests, but the means exist to readily develop the software necessary for such analysis. As shown in Fig. 3.8.2.a-2, the terminal session itself can be saved. An accounting statistics program (COMACNT is the name used here) may be developed to scan the resulting file and count occurrence of key commands.

```
XX.XX.XX SET CONSPPOOL ON
XX.XX.XX XFER CONSPPOOL TO ME
XX.XX.XX
XX.XX.XX (TERMINAL SESSION USING ANY VALID SYSTEM COMMAND)
XX.XX.XX
XX.XX.XX
XX.XX.XX VP CLOSE 9
XX.XX.XX SET CONSPPOOL OFF
XX.XX.XX XFER CONSPPOOL OFF
XX.XX.XX COMACNT
      : (PROGRAM RUNS)
```

Figure 3.8.2.a-2. Collecting and processing statistics for the type and number of requests during a terminal session on a Type II System.

GAERTNER RESEARCH INC

11A140 GAERTNER

GAERTNER RESEARCH INC

ST	DAY	LOGIN	HR	CONNECT	AMT	VPUS	COMPUTE	AMT	UNITS	OTHER	AMT	TOTAL	ACCOUNT	INFO
39	14	14:41	0.67	6.70	116.90	23.38	1000	1.30	31.38	4037				
40	14	15:41	0.99	9.90	125.40	25.08	3800	4.94	39.92	4037				
41	14	16:01	0.00	0.00	0.00	0.00	0	0.50	.50	14160128	PRINT			
42	14	16:30	0.06	0.60	0.00	0.00	0	0.00	.60	TAPE	0282			
43	14	16:39	0.00	0.00	0.00	0.00	0	0.50	.50	14163953	PRINT			
44	15	9:03	0.12	1.20	0.10	0.02	100	0.13	1.35	4037				
45	15	9:10	0.61	6.10	135.20	27.04	6400	8.32	41.46	4037				
46	15	10:56	0.49	4.90	160.10	32.02	7000	9.10	46.02	4037				
47	15	13:06	0.37	3.70	89.10	17.82	3100	4.03	25.55	4037				
48	15	13:33	0.03	0.30	0.40	0.08	100	0.13	.51	4037				
49	15	13:56	0.64	6.40	10.50	2.10	200	0.26	8.76	4037				
50	15	14:45	0.60	6.00	134.00	26.80	4500	5.85	38.65	4037				
51	15	14:53	0.14	1.40	0.00	0.00	0	0.00	1.40	TAPE	0282			
52	15	16:02	0.41	4.10	100.40	20.08	3900	5.07	29.25	4037				
53	15	16:55	0.30	3.00	110.10	22.02	4000	5.20	30.22	4037				
					\$167.74	\$340.56		\$69.04	\$577.34					

DATE	DAYS	CYLINDERS	AMOUNT
01/01/75	15	4	40.89
	4 RENTED TAPE(S) AT \$5.00 PER MONTH		9.75
			50.64

Figure 3.8.2.a-1. Time-share allocation system used by W. W. Gaertner Research, Inc. on an IBM 370/168, a Type II system.

Example 3.8.2.a.3: Terminal Use Statistics Reporting in
a Type III System.

This system also offers certain basic statistics reporting for billing purposes, as illustrated in Fig. 3.8.2.a-3. However, request types are not included in this reporting, nor does this system offer the means to record the terminal session other than the normal printout from a typewriter terminal, if used. Since analysis of sessions is highly system dependent, only the computer manufacturer or some one equally familiar with the system could develop the software to perform this required function.

6000 COST REPORT-PICATINNY ARSENAL

REPORT COVERING MONTH OF JAN 1976
WITH A PRIME/LOW-HOURLY RATE OF \$ 367.20/\$ 244.80

BILL TO FT. MONMOUTH - COST CENTER MGR - GAERTNER RESEARCH INC.

MACHINE	PROGRAM NUMBER	TYPE	JOB NAME OR PF ID	REQUESTERS (OR PF) NAME / COMMENTS	DATE	CHARGE CODE	CDC 65/6600 TIME (MINS)	COST	TOTAL FOR EACH CHARGE CODE
6600			T8T8118	GGGG	1/ 5/76	333	2.46	\$ 15.06	
6600			T8T8118	GGGG	1/ 5/76	333	1.37	\$ 8.38	
6600			T8T8118	GGGG	1/ 5/76	333	1.34	\$ 8.20	
6600	27661	P	GAER017	SCHREYER	1/ 6/76	333	.20	\$ 1.22	
6600	27661	P	GAER012	SCHREYER	1/ 6/76	333	.24	\$ 1.47	
6600	27661	P	GAER017	SCHREYER	1/ 6/76	333	.24	\$ 1.47	
6600	27661	P	GAER02A	SCHREYER	1/ 6/76	333	.20	\$ 1.22	
6600	27661	P	GAER059	SCHREYER	1/ 6/76	333	.69	\$ 4.22	
6600	27661	P	GAER08Y	SCHREYER	1/ 6/76	333	.48	\$ 2.94	
6600	27661	P	GAER09U	SCHREYER	1/ 6/76	333	.49	\$ 3.00	
6600			T8T8118	GGGG	1/ 6/76	333	3.83	\$ 23.44	
6600			T8T8118	GGGG	1/ 6/76	333	3.33	\$ 20.38	
6600			T8T8118	GGGG	1/ 6/76	333	1.60	\$ 9.79	
6600			T8T8118	GGGG	1/ 6/76	333	.14	\$.86	
6600			T8T8118	GGGG	1/ 6/76	333	.05	\$.31	
6600	27661	P	GAER012	SCHREYER	1/ 9/76	333	.24	\$ 1.47	
6600	27661	P	GAER02B	SCHREYER	1/ 9/76	333	1.63	\$ 9.98	
6600	27661	P	GAER02K	SCHREYER	1/ 9/76	333	1.75	\$ 10.71	
6600	27661	P	GAER02T	SCHREYER	1/ 9/76	333	.01	\$.06	
6600			T8T8118	GGGG	1/ 9/76	333	1.68	\$ 10.28	
6600			T8T8118	GGGG	1/ 9/76	333	.05	\$.31	
6600	27661	P	GAER01H	SCHREYER	1/14/76	333	.25	\$ 1.53	
6600	27661	P	GAER01U	SCHREYER	1/14/76	333	1.61	\$ 9.85	
6600	27661	P	GAER017	SCHREYER	1/14/76	333	.02	\$.12	
6600	27661	P	GAER02H	SCHREYER	1/14/76	333	1.75	\$ 10.71	
6600	27661	P	GAER03E	SCHREYER	1/14/76	333	.01	\$.06	
6600	27661	P	GAER09E	SCHREYER	1/14/76	333	.25	\$ 1.02	
6600	27661	P	GAER09G	SCHREYER	1/14/76	333	.01	\$.04	
6600			T8T8118	GGGG	1/14/76	333	.69	\$ 4.22	
6600			T8T8118	GGGG	1/14/76	333	.70	\$ 4.28	
6600	27661	P	GAER01W	SCHREYER	1/14/76	333	.33	\$ 2.02	
6600	27661	P	GAER010	SCHREYER	1/15/76	333	1.61	\$ 9.85	
6600			T8T8118	GGGG	1/15/76	333	.01	\$.06	
6600	27661	P	GAER0AP	SCHREYER	1/15/76	333	1.53	\$ 9.36	
6600					1/26/76	333	.06	\$.37	

Figure 3.8.2.a-3. Collecting and reporting terminal time on a Type III system.

3.8.2.b

Classification: Full

Requirement:

Full on-line implementation requires the capability to restrict the access to specific PSL data files to designated on-line terminals. I.e., access to certain files is allowed only from specific on-line terminals which may be located in physically secure or restricted areas.

Purpose of Paragraph:

To specify the need to be able to limit file access to physically secure terminals.

Examples:

Examples of data security were given in the section of Chapter 5 which covered SPS V-3.5.2.6. As was mentioned there, only operating systems designed for multi-user minicomputers offer this capability. In those systems, file and/or library access can be limited to specific terminals.

Many of the commercial time-sharing systems also can define specific terminal addresses so that only certain user ID's can log on. The same is provided for many Type III systems.

For further illustrations of data security, the reader is advised to reference Chapter 5 of this report.

3.8.2.c

Classification: Full

Requirement:

It is necessary to require some form of user identification and authentication for access to a PSL from any on-line terminal.

Purpose of Paragraph:

To promote data security by access limitation.

Examples:

Several examples of password-keyed access limitations for all three types of systems were given in the discussions concerning SPS V 3.5.2.b.1 and 3.5.2.b.2. The user is advised to reference those sections of this report.

3.8.2.d

Classification: Full

Requirement:

Full on-line implementation requires the existence of a user support facility which allows a PSL user to receive at his terminal a summary of PSL functions and commands. The HELP file allows the on-line user to learn proper utilization of the PSL.

Purpose of Paragraph:

To define a facility which allows a user to become familiar with the interactive commands available.

Examples:

The Type I systems studied do not offer the facility described in this paragraph. Additional software would be required to fulfill this requirement. However, the other two system types do include commands which cause printing of a function summary.

Example 3.8.2.d.1: On-line printing of command summary in a Type II system.

Not all commercial time-sharing systems offer this facility. However, where implemented, it is generally similar to that discussed below.

The CSS implementation allows the user to request information in specific commands (either in brief or detail), on all commands, and on use of the HELP facility itself. The latter is accomplished simply by typing "HELP". "HELP INDEX" causes printing of an alphabetical list of all commands for which documentation exists, while "HELP ALL" prints that documentation for all commands.

Figure 3.8.2.d-1 shows a typical dialogue of HELP use. First, the user simply requests help on the ERASE command. Note that the default is the BRIEF mode, but if DETAIL had been specified following the command name, a more in-depth description would have been given. Next a specific error number and the option "ERROR" are given following the command name, to determine the meaning of that error for the command. In order to find out the meaning of a command option, the word "OPTION" and the name of the option are given following the command name. Finally, a printout of all available commands and their descriptions, in alphabetical order, results from "HELP ALL PRINT".

20.51.46 >help erase

COMMAND ERASE

The ERASE command deletes a file or a related group of files from any writable disk(s). It can also be used to temporarily prevent access to files on a read-only disk.

ERASE filename filetype filemode
ERA * * *

20.52.14 >help erase error 2

000002 The file specified was not found in the user's file directory.

20.52.29 >help load option nomap

NOMAP do not create the file LOAD MAP.

20.52.48 >

CSS

Figure 3.8.2.d-1. Use of HELP facility in a Type II system.

Example 3.8.2.d.2: On-line printing of command summary in a Type III System.

The Type III System studied also offers a command which allows the user to determine the system functions. This is shown in Fig. 3.8.2.d-2. There the user enters the TEACH command and receives a listing of the five areas on which he may receive information. By replying "1" the user then elicits a general description of how to use INTERCOM, the interactive command facility. The subsequent response of "GO" causes a listing of available commands, not in alphabetical order. Each of these commands can be described by typing its number. Note that the command description is highly detailed, and there is no way to request a shorter description.

COMMAND- TEACH
 IF YOU WANT TO KNOW-
 HOW TO USE INTERCOM
 HOW TO USE THE TERMINAL
 AN INTERACTIVE COMMAND
 A SPECIFIC EDITOR COMMAND
 A REMOTE BATCH COMMAND
 - 1
 COMMAND MODE AND DESCRIPTIONS OF COMMANDS-
 THE FUNDAMENTAL ENTRY IN INTERCOM IS A COMMAND.
 THE SYSTEM INDICATES ITS READINESS WITH THE REMOTE
 TERMINAL DISPLAY-
 COMMAND-
 THE USER CAN THEN TYPE THE DESIRED COMMAND, FOLLOWED BY A
 DEPRESSION OF THE SEND KEY (CRT) OR THE RETURN KEY (TTY).
 INTERCOM COMMANDS ARE AS FOLLOWS-
 TO CONTINUE TYPE- GO. TO END TYPE- END.
 -GO
 . INTERCOM COMMANDS--TYPE CORRESPONDING NUMBER

1 = LOGIN	11 = CONNECT	21 = STORE
2 = LOGOUT	12 = DISCONT	22 = FETCH
3 = FILES	13 = XEQ	23 = DISCARD
4 = SITUATE	14 = EDITOR	24 = SAVEFL
5 = SEND	15 = CONVERT	25 = REDUCE
6 = PAGE	16 = BATCH	26 = MAP
7 = ERRORS	17 = Q	27 = SWITCH
8 = ASSETS	18 = M	28 = SCREEN
9 = UTL	19 = LOCK	29 = TEACH
10 = EFL	20 = BRESEQ	30 = CONTINUE

 -2
 * LOGOUT
 TO TERMINATE AN INTERCOME SESSION ENTER
 LOGOUT
 WHEN THIS COMMAND IS ENTERED THE USERS LOCAL AND ATTACHED
 PERMANENT FILES ARE RETURNED. ANY REMOTE FILES ARE RETAINED
 IN THE SYSTEM AWAITING A SUBSEQUENT LOGIN. SESSION ACCOUNTING
 INFORMATION IS RETURNED TO THE TERMINAL.
 TO CONTINUE TYPE- GO. TO END TYPE- END.
 -END
 COMMAND-

Figure 3.8.2.d-2. Use of HELP facility in a Type III system.

3.8.3 Summary of On-Line Terminal Capabilities

The majority of commercially available systems either already offer or can have (with purchasable hardware and software) facilities to fulfill the requirements for this section. The Type II systems are usually designed to be interactive, and so offer the best standard configuration for on-line implementation of a PSL. While Type I systems are also primarily interactive, some are so small that not all the requirements are applicable. Type III systems are not designed for interactive use, and so are the most awkward on which to complement an on line PSL. However, it is still thoroughly possible to do so.

There are definite advantages to an interactive environment, not the least of which is reduced programming/error detection time and a more cost efficient use of human resources. Prospective contractors are urged to consider the feasibility of an on-line PSL, as it will eventually lead to overall reduced costs.

3.9 General Requirements

Requirement:

General support provided by a PSL includes several capabilities which are not directly associated with one of the other major functional areas.

Purpose of Paragraph:

To introduce the section on General Requirements.

3.9.1 Basic General Requirements

Classification: Basic

Requirement:

There are no general support requirements included in the PSL basic requirements.

Purpose of Paragraph:

To state that there are no Basic level General Requirements.

3.9.2 Full General Requirements

Classification: Full

Requirement:

A Full PSL must support the following general requirements.

Purpose of Paragraph:

To introduce the Full General Requirements.

3.9.2.a Concatenation of Data Files

Classification: Full

Requirement:

Concatenated Data Files - A Full PSL implementation requires that the user be able to specify more than one file in all jobs requiring access to a data file. Said multiple files should be concatenated in user-specified order and treated as

though they were one contiguous file. This capability allows the user to combine different versions of files (e.g., development and operational) for the purpose of testing new capabilities error corrections, etc.

Purpose of Paragraph:

To allow a concatenated file to be created for special jobs.

Examples:

Example 3.9.2.a.1: Concatenating data files in a Type I System.

Both Type I systems investigated allow the user to concatenate files. The method of concatenation is the same as that for merging files, since the merge discussed in Example 3.1.2.b.4.1 was not a collating merge. The user is advised to reference that example in Chapter 2 of this report.

Example 3.9.2.a.2: Concatenating data files in a Type II System.

The COMBINE command in CSS and COPYFILE command in CMS perform the concatenation function for the two representative commercial time-sharing systems. COMBINE requires that concatenated files have either all fixed or all variable length records; if fixed, all records should be the same length. An example may be found in Example 3.1.2.b.4.2 in Chapter 2 of this report.

Example 3.9.2.a.3: Concatenating data files in a Type III System.

The representative Type III system allows concatenation by copying all source files into a single destination file. In order to do this, each file must be attached and individually copied into the destination file. The resultant file must then be edited to remove unwanted end-of-file delimiters. An example may be found in Figure 3.1.2.b.4-3.

3.9.2.b Subroutine Support for User Programs

Classification: Full

Requirement:

Subroutine support for user programs - Full PSL implementation requires that programs and/or subroutines used to perform certain PSL functions be made available to the user. Such pro-

grams include index searches, file listings, and other basic tasks which may be needed for integration into software written by the user for his own special needs.

Purpose of Paragraph:

To specify the necessity for user availability of basic PSL function-related programs.

Examples:

The ability of any system to meet this requirement is dependent upon how the basic PSL functions are initially performed: By standard software or by additional software. If the former, compliance with this requirement is usually automatically met, since any special user programs can draw upon the standard system software at need. If the latter, then once the additional software has been supplied it could quite easily be used by the user to supplement his own programs. Assuming that the object code for the PSL programs resides in a specific library which is accessible on a read-level to the user, then the routines may be integrated into the user's programs via standard linkage commands: LINK in the DEC Type I system, LOAD and USE in the commercial time-sharing systems, and COPY in the CDC system. Examples 3.3.1.b.1, -2 and -3 in Chapter 4 of this report further discuss subroutine and co-routine linkage.

3.9.3 Summary of General Requirements

The General Requirements covered in SPS V Section 3.9 are capable of being met by the majority of commercially available systems of all types. It should be noted that file merging as specified in SPS V 3.1.2.4.b can be performed on intra-library as well as inter-library files, but that in neither case is it a collating merge. Rather it is a merge in which one file is appended to the other, i.e., a concatenation. On that basis, Paragraph 3.8.2.a is redundant.

4. SUMMARY AND CONCLUSIONS

The requirements for implementing a PSL fall into two categories: Basic and Full. Generally, the Basic requirements can be fulfilled by any computer system with little or no additional software. Full PSL implementation requires much additional software in any system. Since the two modes of PSL implementation are so different their respective summaries will be given separately.

4.1 Basic PSL Implementation

Table 4.1-1 contains a summary of the Basic PSL requirements and the capabilities of the three system types to fulfill those requirements. Of the systems studied, only the commercial time-sharing systems allowed Basic PSL implementation without additional software. System Types I and III would both require modifications and/or additions. Of course, none provided a PSL installation procedure ready-made. Such an item would have to be supplied or developed on site.

4.1.1 Type I System Summary

Type I systems offer wide variety, but can be basically divided into single-user and multi-user systems. The nature of a PSL is such that it assumes the existence and necessity of a multi-user system. Small operating systems such as RT-11 therefore are not well suited to PSL implementation, although they can fulfill the majority of the requirements. Larger scale operating systems such as the Hewlett-Packard RTE-IV or DEC's RSTS are better suited to PSL implementation in that they are designed for a multi-user environment. Furthermore, their scope is such that they can be easily upgraded toward Full PSL implementation, i.e., they are capable of supporting a PSL level between Basic and Full without much additional software.

4.1.2 Type II System Summary

The commercial time-sharing systems offer the best existing facilities for PSL implementation. They can readily fulfill all Basic requirements, and include system software (EXEC file commands) which allow the user to upgrade the systems very easily. Of the three systems, they are the only type which can support a Full PSL without extensive additions and alterations, as well as being the only type that currently provides support for all Basic standards. Moreover, the system is relatively easy to use, and usually requires fewer individual operations to perform a given task than do the other system types.

Paragraph Number	Requirement	Type I System	Type II System	Type III System
3.1.1.a	Storage of different types of files	OK	OK	OK
3.1.1.b	On-line data access	OK	OK	OK
3.1.1.c	Backup of PSL data	OK	OK	OK though awkward
3.1.1.d	Generation and updating of files	RT-11 cannot generate sequence numbers. OK otherwise.	OK	OK though awkward
3.1.2.a	Compressed storage of files	OK	OK	Not offered by CDC SCOPE
3.2.1.a	Directory listings and information on physical storage of files	OK	OK	CDC SCOPE partially fulfills requirement
3.2.1.b	Full & partial file listings	OK	OK	OK

Table 4.1-1. Summary of Required Basic Capabilities of Three System Types.

Paragraph Number	Requirement	Type I System	Type II System	Type III System
3.2.1.c	Control of data printed on listings	RT-11 only partially fulfills requirement	OK	OK but requires extra user work
3.3.1.a	Syntax checking and compilation	OK	OK	OK
3.3.1.b	Generating program load modules	OK	OK	OK
3.4.1.a	Support of PSL installation	Operating system dependent - programs and/or guidelines to be supplied by Government		
3.4.1.b	Maintenance support of physical storage of data and directories	OK	OK	OK
3.4.1.c	PSL termination	OK	OK	OK
3.5.1	Data security through backup (see 3.1.1.c) and protection in updating	OK	OK	OK

Table 4.1-1 (Continued). Summary of Required Basic Capabilities of Three System Types.

4.1.3 Type III Systems Summary

While the Type III system examined was inferior to the other two in that it was more awkward to work with, it still provided the means to fulfill most of the Basic PSL requirements. The only standard which it could not meet at all was that concerning the compression of files for more efficient storage. Generally, additional software would be required to raise this type of system up to even a very Basic PSL performance level, but such additions would not be prohibitively costly.

4.2 Full PSL Implementation

A Full PSL implementation includes all Basic capabilities plus a several additional features such as Management Data collecting and high security precautions. No currently existing system of any type offers the means to fulfill all these requirements without a considerable amount of additional software. Specifications for some of the additional material have been included in this report, and serve to emphasize the scope of alteration that would be necessary. Even the Type II systems would require major additions. Moreover, commercial time sharing systems are inherently unable to provide sufficient data security for highly classified information, and so must be ruled out on that basis.

In light of these considerations, it is suggested that the Government not require its contractors in the near future to comply with all the standards of Full PSL implementation. Rather those Full level standards which are highly applicable to a given project may be specified in the contract as being necessary; this would allow an intermediate implementation level to be developed which could be tailored to the needs of the job and the existing facilities of the prospective contractor.

The various features which form the Full PSL implementation are interrelated to the extent that they would be most efficiently realized with a comprehensive and integrated software package. If it is considered necessary to require all Full PSL capabilities for a given contract, the Government should supply such a package to the contractor. The development of the package would undoubtedly be both costly and time consuming, too much so to be done individually by Government contractors. It would require extensive human and machine resources.

The integrated PSL package would probably take the form of a pseudo-operating system which would be operationally standardized, i.e., all users would appear to be using the same system even though the supporting computers were different. This would require that there be a set of interface modules for each separate operating system that is to form the basis of a PSL. These interface

modules would possibly be developed in the field by the contractor for an extra consideration. Once developed, other contractors with the same system could access these module sets. Eventually, a standardized structured programming system could be developed that could be implemented on any system.